

Community Experience Distilled

Dominando Python Forensics

Domina el arte de la ciencia forense digital y análisis con Python

Dr. Michael Spreitzenbarth
Dr. Johann Uhrmann

[PACKT] open source*
PUBLISHING
community experience distilled

Dominando Python Forensics

Domina el arte de la ciencia forense digital y
análisis con Python

Dr. Michael Spreitzenbarth

Dr. Johann Uhrmann

[PACKT] open source*
PUBLISHING
community experience distilled

BIRMINGHAM - MUMBAI

Dominando Python Forensics

Primera publicación: octubre de 2015

Referencia de producción: 1261015

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78398-804-4

www.packtpub.com

Créditos

Autores
Dr. Michael Spreitzenbarth
Dr. Johann Uhrmann

Revisores
Richard Marsden
Puneet Narula
Yves Vandermeer

Editor de la comisión
Kartikey Pandey

Editor de adquisiciones
Sonali Vernekar

Editor de desarrollo de contenido
Shweta Pant

Editor Técnico
Pranil Pathare

Editor de copia
Vibha Shukla

Coordinador del proyecto
Shipra Chawhan

Corrector de pruebas
Safis Editing

Indexador
Mariammal Chettiyar

Coordinador de produccion
Arvindkumar Gupta

Trabajo de portada
Arvindkumar Gupta

Sobre los autores

Dr. Michael Spreitzenbarth Tiene un grado de doctor en ingeniería en seguridad IT de la Universidad de Erlangen-Nuremberg y es un CISSP, así como un GMOB. Ha sido consultor de seguridad de IT en CERT en todo el mundo durante más de tres años y ha trabajado como freelancer en el campo de forensics de teléfonos móviles, análisis de malware y consultoría de seguridad de IT por más de seis años. Desde los últimos cuatro años, ha estado dando charlas y conferencias del campo forense y la seguridad móvil en varias universidades y en el sector privado.

Me gustaría dar las gracias a todos los que me han animado a escribir este libro, especialmente a mi esposa por su gran apoyo. También quisiera agradecer a todos los autores de las herramientas de código abierto usadas, sin su ayuda, este libro no habría sido posible.

Dr. Johann Uhrmann Posee una licenciatura en ciencias de la computación de la Universidad de Ciencias Aplicadas de Landshut y un doctorado en ingeniería de la Universidad de las Fuerzas Armadas Federales de Alemania. Cuenta con más de diez años de experiencia en desarrollo de software, que incluye trabajar para start-ups, investigación institucional y ambiente corporativo. Johann tiene varios años de experiencia en manejo de incidentes IT y de gobierno, enfocándose en entornos Linux y Cloud.

En primer lugar, quisiera darle las gracias a mi esposa, Daniela, por su apoyo moral y su voluntad de renunciar a algún tiempo familiar mientras escribía. También quiero agradecer a mi coautor y colega, el Dr. Michael Spreitzenbarth, por contactarme para escribir este libro y manejar una gran parte de los gastos generales de organización de tal proyecto. Además, la gran gente que trabaja en todos los proyectos de software de código abierto que usamos y mencionamos en este libro merecen crédito. Ustedes son los chicos que mantienen el mundo de las IT girando.

Acerca de los revisores

Richard Marsden Tiene más de veinte años de experiencia profesional en desarrollo de software. Después de comenzar en los campos de la geofísica y la exploración de petróleo, ha pasado los últimos doce años a cargo de Winwaed Software Technology LLC, un proveedor de software independiente. Winwaed se especializa en herramientas geoespaciales y aplicaciones, que incluyen aplicaciones web, y opera el sitio web http://www.mapping-tools.com para herramientas y complementos para productos geoespaciales, como Maptitude de Caliper y MapPoint de Microsoft.

Richard también fue un revisor técnico para *Python Geospatial Development*, y *Python Geospatial Analysis Essentials*, ambos escritos por Erik Westra, *Packt Publishing*.

Puneet Narula Está trabajando actualmente como analista de datos PPC con Hostelworld. com Ltd (`http://www.hostelworld.com/`), en Dublín, Irlanda, donde analiza datos masivos de clickstream de fuentes directas y de afiliados y proporciona información al equipo de marketing digital. Él utiliza RapidMiner, R, y Python para el análisis exploratorio y predictivo. Sus áreas de especialización son la programación en Python y R, aprendizaje de máquinas, análisis de datos y Tableau.

Comenzó su carrera en banca y finanzas y luego se trasladó al dominio cada vez mayor de datos y análisis.

Obtuvo una maestría en computación (análisis de datos) en el Instituto de Tecnología de Dublín, en Dublín, Irlanda. Ha revisado los libros: *Python Data Analysis*, por *Ivan Idris*, *Packt Publishing* y *Python Geospatial Analysis Essentials*, por *Erik Westra*, *Packt Publishing*.

Yves Vandermeer Es un oficial de policía que trabaja para la policía federal belga. Él ha estado implicado en investigaciones importantes desde 1997, donde él contribuyó a la recuperación de la evidencia digital. Posee una maestría en informática forense, Y es también un entrenador en varios temas tales como sistemas de archivos y forense de red para varias agencias de aplicación de la ley.

Presidencia del Grupo Europeo de Formación y Educación sobre Delito Cibernético, E.C.T.E.G., desde 2013, Y apoya la creación de materiales formativos enfocados a la comprensión de los conceptos aplicados en los ejercicios prácticos.

Haciendo uso de su experiencia, ha desarrollado herramientas de software forense para la aplicación de la ley y contribuyó a varios grupos de asesoramiento relacionados con la delincuencia informática y la informática forense.

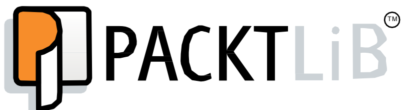
www.PacktPub.com

Archivos de soporte, libros electrónicos, ofertas de descuento, y más

Para obtener soporte y descargas relacionadas con su libro, visite www.PacktPub.com.

¿Sabía usted que Packt ofrece versiones eBook de cada libro publicado, con PDF y ePub disponibles? Puede actualizar a la versión de eBook en www.PacktPub.com y como un cliente de libro de impresión, tiene derecho a un descuento en la copia de eBook. Póngase en contacto con nosotros en service@packtpub.com para más detalles.

En www.PacktPub.com, también puede leer una colección de artículos técnicos gratuitos, suscríbase a una serie de boletines gratuitos y reciba descuentos exclusivos y ofertas sobre libros y libros electrónicos Packt.



`https://www2.packtpub.com/books/subscription/packtlib`

¿Necesita soluciones instantáneas para sus preguntas de IT? PacktLib es la biblioteca digital en línea de Packt. Aquí, puede buscar, acceder y leer toda la librería de libros de Packt.

¿Por qué suscribirse?

- Se puede buscar a través de cada libro publicado por Packt
- Copiar y pegar, imprimir y marcar contenido
- A petición y accesible a través de un navegador web

Acceso gratuito para los titulares de la cuenta Packt

Si tiene una cuenta con Packt en www.PacktPub.com, puede utilizarla para acceder a PacktLib hoy y ver 9 libros totalmente gratuitos. Simplemente utilice sus credenciales de acceso para acceso inmediato.

Tabla de contenido

Prefacio

Hoy en día, la tecnología de la información es parte de casi todo lo que nos rodea. Estos son los sistemas que usamos y que nos apoyan en la construcción y funcionamiento de ciudades, empresas, nuestros tours personales de compras en línea, y nuestras amistades. Estos sistemas son atractivos para el uso y abuso. En consecuencia, todos los campos criminales como el robo, el fraude, el chantaje, etc. se expandieron a la IT. Hoy en día, se trata de una industria de la sombra, multimillonaria, criminal y global.

¿Puede una persona detectar señales de actividades delictivas o sospechosas de una industria de la sombra, multimillonaria, criminal y global.? Bueno, a veces puedes. Para analizar el crimen moderno, usted no necesita lupas y levantamiento de huellas de botellas de vino. En su lugar, vamos a ver cómo aplicar sus habilidades de Python para ver de cerca los puntos más prometedores en un sistema de archivos y tomar huellas dactilares digitales de las huellas dejadas por los hackers.

Como autores, creemos en la fuerza de los ejemplos sobre la teoría polvorienta. Esta es la razón por la que proporcionamos ejemplos de herramientas forenses y scripts, que son lo suficientemente cortas para ser entendidas por el programador de Python promedio, herramientas utilizables y bloques de construcción para el mundo real de la investigación forense.

¿Estás listo para convertir la sospecha en hechos concretos?

Lo que este libro cubre

Capítulo 1, Configuración del laboratorio e introducción a Python ctypes, cubre cómo configurar su entorno para seguir los ejemplos que se proporcionan en este libro. Vamos a echar un vistazo a los diversos módulos de Python que apoyan nuestros análisis forenses. Con ctypes, ofrecemos los medios para ir más allá de los módulos de Python y aprovechar las capacidades de las bibliotecas de sistemas nativos.

Capítulo 2, Algoritmos Forenses, le proporciona el equivalente digital de tomar huellas dactilares. Al igual que en el caso de las huellas dactilares clásicas, le mostraremos cómo comparar las huellas dactilares digitales con un enorme registro de las muestras buenas y malas conocidas. Esto le ayudará a centrar su análisis y proporcionar una prueba de solidez forense.

Capítulo 3, Uso de Python para Windows y Linux Forensics, es el primer paso en su camino hacia la comprensión de la evidencia digital. Proporcionaremos ejemplos para detectar signos de compromiso en sistemas Windows y Linux. Concluiremos el capítulo con un ejemplo sobre cómo utilizar algoritmos de aprendizaje automático en el análisis forense.

Capítulo 4, Uso de Python para análisis forense de redes, se trata de capturar y analizar el tráfico de red. Con las herramientas proporcionadas, puede buscar y analizar el tráfico de la red para detectar signos de infiltración o la firma de la comunicación de malware.

Capítulo 5, Uso de Python para virtualización Forense, explica cómo los conceptos modernos de virtualización pueden ser utilizados por el atacante y el analista forense. En consecuencia, mostraremos cómo detectar comportamientos maliciosos en el nivel del hipervisor y utilizar la capa de virtualización como fuente confiable de datos forenses.

Capítulo 6, Uso de Python para forense móvil, le dará una idea sobre cómo recuperar y analizar datos forenses de dispositivos móviles. Los ejemplos incluirán el análisis de dispositivos Android, así como dispositivos Apple iOS.

Capítulo 7, El uso de Python para el análisis forense de memoria, demuestra cómo recuperar snapshots de memoria y analizar estas imágenes forenses de RAM con Linux y Android. Con la ayuda de herramientas como LiME y Volatility, demostraremos cómo extraer información de la memoria del sistema.

Lo que necesitas para este libro

Todo lo que necesita para este libro es una estación de trabajo Linux con un entorno Python 2.7 y una conexión a Internet que funcione. Capítulo 1, Configuración del laboratorio e introducción a Python ctypes le guiará a través de la instalación de los módulos y herramientas adicionales de Python. Todas nuestras herramientas usadas están disponibles gratuitamente en Internet. El código fuente de nuestras muestras está disponible en Packt Publishing.

Para seguir los ejemplos del Capítulo 5, Uso de Python para virtualización Forense, puede configurar un entorno de virtualización con VMware vSphere. El software necesario está disponible en VMware como versión de prueba limitada en el tiempo sin restricciones funcionales.

Aunque no es estrictamente requerido, recomendamos probar algunos de los ejemplos del Capítulo 6, Uso de Python para forense móvil, en dispositivos móviles descartados. Para sus primeros experimentos, por favor, abstenerse de utilizar teléfonos personales o empresariales que estén realmente en uso.

¿Para quién es este libro?

Este libro es para administradores de IT, operaciones de IT y analistas que desean adquirir profundas habilidades en la recopilación y análisis de evidencia digital. Si ya eres un experto forense, este libro te ayudará a ampliar tus conocimientos en nuevas áreas, como la virtualización o los dispositivos móviles.

Para sacar el máximo provecho de este libro, debe tener habilidades decentes en Python y comprender al menos algunos funcionamientos internos de sus objetivos forenses. Por ejemplo, algunos detalles del sistema de archivos.

Convenciones

En este libro, encontrará una serie de estilos de texto que distinguen entre diferentes tipos de información. Aquí hay algunos ejemplos de estos estilos y una explicación de su significado.

Las palabras de código en el texto, los nombres de las tablas de base de datos, los nombres de las carpetas, los nombres de archivo, las extensiones de archivo, los nombres de ruta, las direcciones URL falsas, la entrada de usuario y las descripciones de Twitter se muestran como sigue: "Note que en el caso de Windows, `msvcrt` es la biblioteca estándar C de MS que contiene la mayoría de las funciones estándar de C y utiliza la convención de llamada `cdecl` (en sistemas Linux, la librería similar sería `libc.so.6`)."

Un bloque de código se establece de la siguiente manera:

```
def multi_hash(filename):
    """Calculates the md5 and sha256 hashes
       of the specified file and returns a list
       containing the hash sums as hex strings."""
```

Cuando queremos llamar su atención sobre una parte en particular de un bloque de código, las líneas o elementos relevantes se definen en negrita:

```
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/
event"><System><Provider Name="Microsoft-Windows-Security-
Auditing" Guid="54849625-5478-4994-a5ba-3e3b0328c30d"></Provider>
<EventID Qualifiers="">4724</EventID>
<Version>0</Version>
<Level>0</Level>
<Task>13824</Task>
```

Cualquier entrada de línea de comandos o salida se escribe de la siguiente manera:

```
user@lab:~$ virtualenv labenv
New python executable in labenv/bin/python
Installing setuptools, pip...done.
```

Los nuevos términos y **palabras importantes** ase muestran en negrita. Las palabras que se ven en la pantalla, por ejemplo, en menús o cuadros de diálogo, aparecen en el texto como este: "Cuando se le pregunte a **Select System Logs**, asegúrese de que todos los tipos de registro estén seleccionados."



Las advertencias o notas importantes aparecen en un cuadro como este.



Consejos y trucos aparecen así.

Comentarios del lector

Los comentarios de nuestros lectores es siempre bienvenido. Háganos saber lo que piensa acerca de este libro, lo que le gustó o no le gustó. La retroalimentación del lector es importante para nosotros, ya que nos ayuda a desarrollar títulos de los que realmente sacarás el máximo provecho.

Para enviarnos comentarios generales, simplemente envíe un correo electrónico a feedback@packtpub.com, y mencione el título del libro en el tema de su mensaje.

Si hay un tema en el que tiene experiencia y está interesado en escribir o contribuir a un libro, consulte nuestra guía de autores en www.packtpub.com/authors.

Atención al cliente

Ahora que usted es el orgulloso propietario de un libro de Packt, tenemos una serie de cosas que le ayudarán a sacar el máximo provecho de su compra.

Descargando el código de ejemplo

Puede descargar los archivos de código de ejemplo de su cuenta en http://www.packtpub.com para todos los libros de Packt Publishing que haya comprado. Si ha comprado este libro en otro lugar, puede visitar http://www.packtpub.com/support y registrarse para que los archivos se envíen por correo electrónico directamente a usted.

Errata

Aunque hemos tomado todas las precauciones para garantizar la exactitud de nuestro contenido, los errores se producen. Si encuentras un error en uno de nuestros libros -tal vez un error en el texto o el código- estaríamos agradecidos si pudieras informarnos esto. Al hacerlo, puede salvar a otros lectores de la frustración y ayudarnos a mejorar las versiones posteriores de este libro. Si encuentra alguna errata, infórmenos visitando http://www.packtpub. com/submit-errata, seleccionando su libro, haciendo clic en el enlace **Formulario de envío de erratas** e ingresando los detalles de sus erratas. Una vez que se hayan verificado sus erratas, se aceptará su envío y las erratas se cargarán en nuestro sitio web o se agregarán a cualquier lista de erratas existentes en la sección Erratas de ese título

Para ver las erratas presentadas anteriormente, vaya a https://www.packtpub.com/books/content/support e introduzca el nombre del libro en el campo de búsqueda. La información requerida aparecerá en la sección **Errata**.

Piratería

La piratería de material protegido por derechos de autor en Internet es un problema continuo en todos los medios de comunicación. En Packt, tomamos muy en serio la protección de nuestros derechos de autor y licencias. Si encuentra copias ilegales de nuestras obras en cualquier forma en Internet, por favor proporcione la dirección de la ubicación o el nombre del sitio web inmediatamente para que podamos buscar un remedio.

Por favor, póngase en contacto con nosotros en copyright@packtpub.com con un enlace a la sospecha de material pirateado.

Agradecemos su ayuda en la protección de nuestros autores y nuestra capacidad para ofrecerle contenido valioso.

Preguntas

Si tiene algún problema con cualquier aspecto de este libro, puede ponerse en contacto con nosotros en questions@packtpub.com, y haremos todo lo posible para resolver el problema.

1

Configuración del laboratorio e Introducción a Python ctypes

Cyber Security y **Digital Forensics** son dos temas de creciente importancia. La investigación forense digital, en particular, está cobrando cada vez más importancia, no sólo durante las investigaciones policiales, sino también en el campo de la respuesta a incidentes. Durante todas las investigaciones mencionadas anteriormente, es fundamental conocer la causa raíz de una violación de seguridad, un mal funcionamiento de un sistema o un crimen. La investigación forense digital juega un papel importante en la superación de estos desafíos.

En este libro, le enseñaremos cómo construir su propio laboratorio y realizar profundas investigaciones forenses digitales, que se originan en una amplia gama de plataformas y sistemas, con la ayuda de Python. Comenzaremos con las máquinas de escritorio comunes de Windows y Linux, luego avanzaremos hacia las plataformas de cloud y virtualización y acabaremos con los teléfonos móviles. No sólo le mostraremos cómo examinar los datos en reposo o en tránsito, sino que también echaremos un vistazo más profundo a la memoria volátil.

Python proporciona una excelente plataforma de desarrollo para crear sus propias herramientas de investigación debido a su menor complejidad, mayor eficiencia, gran número de bibliotecas de terceros y también es fácil de leer y escribir. Durante el viaje de lectura de este libro, no sólo aprenderá a utilizar las bibliotecas y extensiones más comunes de Python para analizar la evidencia, sino también cómo escribir sus propios scripts y herramientas de ayuda para trabajar más rápido en los casos o incidentes con una enorme cantidad de evidencia que tiene que ser analizada. Comencemos nuestro viaje para dominar Python forensics estableciendo nuestro ambiente del laboratorio, seguido por una breve introducción de los ctypes de Python.

Si ya ha trabajado con **ctypes** de Python y tiene un entorno de laboratorio de trabajo, no dude en saltar el primer capítulo y comenzar directamente con uno de los otros capítulos. Después del primer capítulo, los otros capítulos son bastante independientes entre sí y se pueden leer en cualquier orden.

Configuración del laboratorio

Como base para nuestros scripts e investigaciones, necesitamos un entorno de laboratorio completo y poderoso que sea capaz de manejar un gran número de diferentes tipos de archivos y estructuras, así como conexiones a dispositivos móviles. Para lograr este objetivo, utilizaremos la última versión 14.04.2 de Ubuntu LTS e instalaremos en una máquina virtual (VM). En las secciones siguientes, explicaremos la configuración de la máquina virtual e introduciremos **virtualenv** de Python, que utilizaremos para establecer nuestro entorno de trabajo.

Ubuntu

Para trabajar en un entorno de laboratorio similar, le sugerimos que descargue una copia de la última Ubuntu LTS Desktop Distribution de http://www.ubuntu.com/download/desktop/, preferiblemente la versión de 32 bits. La distribución proporcionauna interfaz de usuario fácil de usar y ya tiene el entorno de Python 2.7.6 instalado y preconfigurado. A lo largo del libro, usaremos Python 2.7.x y no las versiones 3.x más recientes. Varios ejemplos y estudios de caso en este libro dependerán de las herramientas o bibliotecas que ya forman parte de la distribución de Ubuntu. Cuando un capítulo o una sección del libro requiera un paquete o biblioteca de terceros, proporcionaremos la información adicional sobre cómo instalarlo en el **virtualenv** (la configuración de este entorno se explicará en la siguiente sección) o en Ubuntu en general .

Para un mejor rendimiento del sistema, recomendamos que la máquina virtual que se utiliza para el laboratorio tenga al menos 4 GB de memoria volátil y unos 40 GB de almacenamiento.

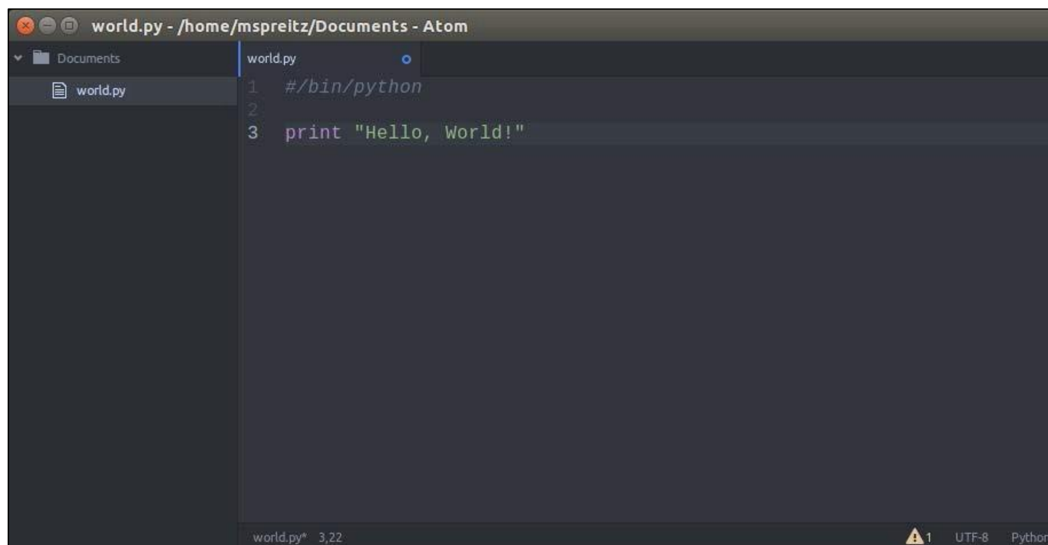


Figure 1: The Atom editor

Para escribir su primer script de Python, puede usar un editor simple como **vi** o un IDE potente pero abarrotado como **eclipse**. Como una alternativa realmente potente, te sugerimos que uses **atom**, un editor muy limpio pero altamente personalizable que se puede descargar libremente desde https://atom.io/

Entorno virtual de Python (virtualenv)

De acuerdo con la documentación oficial de Python, Virtual Environment es una herramienta para mantener las dependencias requeridas por diferentes proyectos en lugares separados creando entornos Python virtuales para ellos. Resuelve el dilema "Proyecto X depende de la versión 1.x, pero el Proyecto Y necesita 4.x" y mantiene su directorio global de sitios-paquetes limpio y manejable.

Esto es también lo que vamos a utilizar en los siguientes capítulos para mantener un entorno común para todos los lectores del libro y no ejecutar cualquier problema de compatibilidad. En primer lugar, tenemos que instalar el paquete **virtualenv**. Esto se hace con el siguiente comando:

```
user@lab:~$ pip install virtualenv
```

Ahora crearemos una carpeta en el directorio personal de los usuarios para nuestro entorno Python virtual. Este directorio contendrá los archivos ejecutables de Python y una copia de la biblioteca de pip, que se puede utilizar para instalar otros paquetes en el entorno. El nombre del entorno virtual (en nuestro caso, se llama **labenv**) puede ser de su elección. Nuestro entorno de laboratorio virtual se puede crear ejecutando el siguiente comando:

```
user@lab:~$ virtualenv labenv
New python executable in labenv/bin/python
Installing setuptools, pip...done.
```

Para empezar a trabajar con el nuevo entorno de laboratorio, primero debe activarse. Esto se puede hacer a través de:

```
user@lab:~$ source labenv/bin/activate
(labenv)user@lab:~$
```

Ahora, puede ver que el símbolo del sistema comienza con el nombre del entorno virtual que activamos. De ahora en adelante, cualquier paquete que instales usando pip será colocado en la carpeta **labenv**, aislada de la instalación global de Python en el Ubuntu subyacente.

A lo largo del libro, utilizaremos este entorno virtual de python e instalaremos nuevos paquetes y bibliotecas en él de vez en cuando. Por lo tanto, cada vez que intente recapitular un ejemplo mostrado recuerde o desafíe a cambiar al entorno **labenv** antes de ejecutar sus scripts.

Si ha terminado de trabajar en el entorno virtual por el momento y desea volver a su entorno "normal" de Python, puede desactivar el entorno virtual ejecutando el siguiente comando:

```
(labenv)user@lab:~$ deactivate
user@lab:~$
```

Esto lo pone de nuevo en el intérprete de Python predeterminado del sistema con todas sus bibliotecas y dependencias instaladas.

Si está utilizando más de una máquina virtual o física para las investigaciones, los entornos virtuales pueden ayudarle a mantener sus bibliotecas y paquetes sincronizados con todos estos lugares de trabajo. Con el fin de asegurar que sus entornos sean consistentes, es una buena idea "congelar" el estado actual de los paquetes de entorno. Para ello, ejecute:

```
(labenv)user@lab:~$ pip freeze > requirenments.txt
```

Esto creará un archivo requirements.txt, que contiene una lista simple de todos los paquetes en el entorno actual y sus respectivas versiones. Si desea instalar los mismos paquetes utilizando la misma versión en una máquina diferente, simplemente copie el archivo requirements.txt en la máquina deseada, cree el entorno **labenv** como se describió anteriormente y ejecute el siguiente comando:

```
(labenv)user@lab:~$ pip install -r requirements.txt
```

Ahora, tendrá entornos Python consistentes en todas las máquinas y no tendrá que preocuparse por las diferentes versiones de la biblioteca u otras dependencias.

Después de haber creado la máquina virtual Ubuntu con nuestro ambiente de laboratorio dedicado, estamos casi listos para comenzar nuestro primer análisis forense. Pero antes de eso, necesitamos más conocimiento de las bibliotecas y fondos útiles de Python. Por lo tanto, comenzaremos con una introducción a Python **ctypes** en la siguiente sección.

Introducción a Python ctypes

De acuerdo con la documentación oficial de Python, **ctypes** es una biblioteca de funciones externa que proporciona tipos de datos compatibles con C y permite llamar a funciones en DLL o bibliotecas compartidas. Una biblioteca de funciones externa significa que el código Python puede llamar a funciones C utilizando sólo Python, sin necesidad de extensiones especiales o personalizadas.

Este módulo es una de las bibliotecas más poderosas disponibles para el desarrollador de Python. La biblioteca **ctypes** le permite no sólo llamar a funciones en bibliotecas enlazadas dinámicamente (como se describió anteriormente), sino que también puede utilizarse para la manipulación de memoria de bajo nivel. Es importante que comprenda los conceptos básicos de cómo utilizar la biblioteca **ctypes**, ya que se utilizará para muchos ejemplos y casos reales en todo el libro.

En las siguientes secciones, vamos a introducir algunas características básicas de Ptyon **ctypes** y cómo usarlos.

Trabajar con bibliotecas de vínculos dinámicos

ctypes de Python exporta el cdll y en Windows el windll u objetos oledll respectivamente, para cargar las bibliotecas de vínculos dinámicos solicitadas. Una biblioteca vinculada dinámicamente es un binario compilado que se vincula en tiempo de ejecución al proceso ejecutable principal. En plataformas Windows, estos binarios se llaman **Dynamic Link Libraries** (DLL) y en Linux se denominan **shared objects** (SO). Puede cargar estas bibliotecas enlazadas accediendo a ellas como atributos de los objetos cdll, windll u oledll. Ahora, vamos a demostrar un ejemplo muy breve para Windows y Linux para obtener la hora actual directamente de la función de tiempo en libc (esta biblioteca define las llamadas al sistema y otras facilidades básicas como open, printf o exit).

Observe que en el caso de Windows, `msvcrt` es la biblioteca estándar C de MS que contiene la mayoría de las funciones estándar de C y usa la convención de llamada `cdecl` (en los sistemas Linux, la librería similar sería `libc.so.6`):

```
C:\Users\Admin>python

>>> from ctypes import *
>>> libc = cdll.msvcrt
>>> print libc.time(None)
1428180920
```

Windows agrega automáticamente el sufijo usual del archivo `.dll`. En Linux, se requiere especificar el nombre del archivo, incluyendo la extensión, para cargar la biblioteca elegida. Debe utilizarse el método `LoadLibrary()` de los cargadores DLL o cargar la biblioteca creando una instancia de `CDLL` llamando al constructor, como se muestra en el siguiente código:

```
(labenv)user@lab:~$ python

>>> from ctypes import *
>>> libc = CDLL("libc.so.6")
>>> print libc.time(None)
1428180920
```

Como se muestra en estos dos ejemplos, es muy fácil ser capaz de llamar a una biblioteca dinámica y utilizar una función que se exporta. Usted utilizará esta técnica muchas veces en todo el libro, por lo que es importante que entienda cómo funciona.

Tipos de datos C

Al examinar los dos ejemplos de la sección anterior con detalle, puede ver que usamos `None` como uno de los parámetros de una biblioteca C enlazada dinámicamente. Esto es posible porque `None, integers, longs,byte strings,` y `unicode strings` son los objetos nativos de Python que se pueden usar directamente como parámetros en estas llamadas de función. `None` se pasa como en C, `NULL pointer, byte strings,` y `unicode strings` se pasan como punteros al bloque de memoria que contiene sus datos (char* o wchar_t *). Los `integers` en Python y los `longs` en Python se pasan como el `int type` en C, por defecto de la plataforma, su valor se oculta para encajar en el tipo C. Una descripción completa de los Python types y sus ctype types correspondientes se puede ver en la Tabla 1:

ctypes type	C type	Python type
c_bool (`https://docs.python.org/2/library/ctypes.html#ctypes.c_bool`)	_Bool	bool (1)
c_char (`https://docs.python.org/2/library/ctypes.html#ctypes.c_char`)	char	1-character string
c_wchar (`https://docs.python.org/2/library/ctypes.html#ctypes.c_wchar`)	wchar_t	1-character unicode string
c_byte (`https://docs.python.org/2/library/ctypes.html#ctypes.c_byte`)	char	int/long
c_ubyte (`https://docs.python.org/2/library/ctypes.html#ctypes.c_ubyte`)	unsigned char	int/long
c_short (`https://docs.python.org/2/library/ctypes.html#ctypes.c_short`)	short	int/long
c_ushort (`https://docs.python.org/2/library/ctypes.html#ctypes.c_ushort`)	unsigned short	int/long
c_int (`https://docs.python.org/2/library/ctypes.html#ctypes.c_int`)	int	int/long
c_uint (`https://docs.python.org/2/library/ctypes.html#ctypes.c_uint`)	unsigned int	int/long
c_long (`https://docs.python.org/2/library/ctypes.html#ctypes.c_long`)	long	int/long

ctypes type	C type	Python type
c_ulong (https://docs.python.org/2/library/ctypes.html#ctypes.c_ulong)	unsigned long	int/long
c_longlong (https://docs.python.org/2/library/ctypes.html#ctypes.c_longlong)	__int64 or long long	int/long
c_ulonglong (https://docs.python.org/2/library/ctypes.html#ctypes.c_ulonglong)	unsigned int64 or unsigned long long	int/long
c_float (https://docs.python.org/2/library/ctypes.html#ctypes.c_float)	float	float
c_double (https://docs.python.org/2/library/ctypes.html#ctypes.c_double)	double	float
c_longdouble (https://docs.python.org/2/library/ctypes.html#ctypes.c_longdouble)	long double	float
c_char_p (https://docs.python.org/2/library/ctypes.html#ctypes.c_char_p)	char * (NUL terminated)	string or None
c_wchar_p (https://docs.python.org/2/library/ctypes.html#ctypes.c_wchar_p)	wchar_t * (NUL terminated)	unicode or None
c_void_p (https://docs.python.org/2/library/ctypes.html#ctypes.c_void_p)	void *	int/long or None

Table 1: Fundamental Data Types

Esta tabla es muy útil porque todos los Python types excepto `integers`, `strings`, y `unicode strings` tienen que ser envueltos en sus ctypes type correspondientes para que puedan convertirse en el tipo de datos en C necesario en la biblioteca vinculada y no lanzar las excepciones TypeError, como se muestra en la siguiente código:

```
(labenv)user@lab:~$ python

>>> from ctypes import *
>>> libc = CDLL("libc.so.6")
>>> printf = libc.printf

>>> printf("An int %d, a double %f\n", 4711, 47.11)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ctypes.ArgumentError: argument 3: <type 'exceptions.TypeError'>: Don't
know how to convert parameter 3

>>> printf("An int %d, a double %f\n", 4711, c_double(47.11))
An int 4711, a double 47.110000
```

Definición de uniones y estructuras

Unions y **Structures** son tipos de datos importantes porque se utilizan con frecuencia en toda la `libc` en Linux y también en la API de Microsoft Win32.

Los uniones son simplemente un grupo de variables, que pueden ser del mismo tipo o de tipos de datos diferentes, donde todos sus miembros comparten la misma ubicación de memoria. Al almacenar variables de esta manera, las uniones permiten especificar el mismo valor en diferentes tipos. Para el próximo ejemplo, pasaremos de la shell interactiva de Python al editor de átom en nuestro entorno de laboratorio de Ubuntu. Sólo tiene que abrir el editor de átom, escriba el código siguiente y guárdelo bajo el nombre `new_evidence.py`:

```
from ctypes import *

class case(Union):
        _fields_ = [
        ("evidence_int", c_int),
        ("evidence_long", c_long),
        ("evidence_char", c_char * 4)
        ]

value = raw_input("Enter new evidence number:")
new_evidence = case(int(value))
print "Evidence number as a int: %i" % new_evidence.evidence_int
print "Evidence number as a long: %ld" %
new_evidence.evidence_long print "Evidence number as a char: %s" %
new_evidence.evidence_char
```

Si asigna a la variable miembro `evidence` union `evidence_int` un valor de 42, puede utilizar el miembro evidence_char para mostrar la representación de caracteres de ese número, como se muestra en el ejemplo siguiente:

```
(labenv)user@lab:~$ python new_evidence.py

Enter new evidence number:42

Evidence number as a long: 42
Evidence number as a int: 42
Evidence number as a char: *
```

Como puede ver en el ejemplo anterior, al asignar un valor único a union, obtendrá tres representaciones diferentes de ese valor. Para `int` y `long`, la salida mostrada es obvia, pero para la variable `evidence_char`, podría ser un poco confuso. En este caso, '*' es el carácter ASCII con el valor equivalente decimal a 42. La variable miembro `evidence_char` es un buen ejemplo de cómo definir un `array` en ctypes. En ctypes, un array se define multiplicando un type por el número de elementos que desea asignar en el array. En este ejemplo, se definió un array de caracteres de cuatro elementos para la variable miembro `evidence_char`.

Una estructura es muy similar a las uniones, pero los miembros no comparten la misma ubicación de memoria. Puede acceder a cualquiera de las variables miembro en la estructura utilizando notación de punto, como `case.name`. Esto tendría acceso a la variable de `name` contenida en la estructura del caso. El siguiente es un breve ejemplo de cómo crear una `estructura` (o struct, como se les llama a menudo) con tres miembros: `name`, `number`, e `investigator_name` para que todos puedan acceder mediante la notación de puntos:

```
from ctypes import *

class case(Structure):
        _fields_ = [
        ("name", c_char * 16),
        ("number", c_int),
        ("investigator_name", c_char * 8)
        ]
```



Descargando el código de ejemplo

Puede descargar los archivos de código de ejemplo de su cuenta en http://www.packtpub.com para todos los libros de Packt Publishing que haya comprado. Si ha comprado este libro en otro lugar, puede visitar http://www.packtpub.com/support y registrarse para que los archivos se envíen por correo electrónico directamente a usted.

Resumen

En el primer capítulo, hemos creado nuestro entorno de laboratorio: una máquina virtual con **Ubuntu 14.04.2 LTS**. Este paso es realmente importante, ya que ahora puede crear snapshots antes de trabajar en pruebas reales y son capaces de volver a un estado de máquina limpia después de terminar la investigación. Esto puede ser útil, especialmente, cuando se trabaja con copias de seguridad comprometidas del sistema, en las que desea asegurarse de que su sistema está limpio al trabajar en un caso diferente después.

En la segunda parte de este capítulo, hemos demostrado cómo trabajar con **entornos virtuales de Python (virtualenv)** que se utilizarán y se extenderán a lo largo del libro.

En la última sección de este capítulo, le presentamos Python **ctypes**, una biblioteca muy poderosa disponible para el desarrollador de Python. Con **ctypes**, no sólo puede llamar a funciones en las bibliotecas vinculadas dinámicamente (API de Microsoft Win32 disponibles o objetos compartidos comunes de Linux), sino que también se pueden utilizarla para la manipulación de memoria de bajo nivel.

Después de completar este capítulo, tendrá un entorno básico creado para ser utilizado para el resto del libro, y también comprenderá los fundamentos de Python **ctypes** que serán útiles en algunos de los siguientes capítulos.

2

Algoritmos forenses

Los algoritmos forenses son los bloques de construcción para un investigador forense. Independientemente de cualquier implementación específica, estos algoritmos describen los detalles de los procedimientos forenses. En la primera sección de este capítulo, vamos a introducir los diferentes algoritmos que se utilizan en las investigaciones forenses, incluyendo sus ventajas y desventajas.

Algoritmos

En esta sección, describimos las principales diferencias entre **MD5, SHA256** y **SSDEEP,** los algoritmos más comunes utilizados en las investigaciones forenses. Vamos a explicar los casos de uso, así como las limitaciones y amenazas detrás de estos tres algoritmos. Esto debería ayudarle a entender por qué es mejor usar SHA256 que usar MD5 y en qué casos SSDEEP puede ayudarle en la investigación.

Antes de sumergirnos en las diferentes funciones hash, daremos un breve resumen de lo que es una función hash criptográfica.

Una **función hash** es una función que asigna una cantidad arbitrariamente grande de datos a un valor de una longitud fija. La función hash asegura que la misma entrada siempre da como resultado la misma salida, llamada suma de hash. Por consiguiente, una suma de hash es una característica de una pieza específica de datos.

Una **función hash criptográfica** es una función hash que se considera prácticamente imposible de invertir. Esto significa que no es posible crear los datos de entrada con un valor de suma de hash predefinido por ningún otro medio que intentar todos los valores de entrada posibles, es decir, la fuerza bruta. Por lo tanto, esta clase de algoritmos se conoce como unidireccional algoritmo criptográfico.

La función hash criptográfica ideal tiene cuatro propiedades principales, como las siguentes:

1. Debe ser fácil calcular el valor hash para cualquier entrada dada.
2. Debe ser imposible generar la entrada original de su hash.
3. Debe ser imposible modificar la entrada sin cambiar el hash.
4. Debe ser imposible encontrar dos entradas diferentes con el mismo hash (**resistente a las colisiones**).

En el caso ideal, de crea un hash de entrada dado y cambia sólo un bit de esta entrada, el hash recién calculado se verá totalmente diferente, de la siguiente manera:

```
user@lab:~$ echo -n This is a test message | md5sum
fafb00f5732ab283681e124bf8747ed1

user@lab:~$ echo -n This is A test message | md5sum
aafb38820e0a3788eb41e9f5805e088e
```

Si se cumplen todas las propiedades anteriormente mencionadas, el algoritmo es una función hash criptográficamente correcta y puede utilizarse para comparar, por ejemplo, archivos entre sí para probar que no han sido manipulados durante el análisis o por un atacante.

MD5

El algoritmo de digestión de mensajes MD5 fue la función de hash criptográfica más utilizada (y sigue siendo ampliamente utilizada) produce un valor de hash de 128 bits (16 bytes), expresado típicamente en el formato de texto como un número hexadecimal de 32 dígitos como se muestra en el ejemplo anterior). Este resumen de mensajes se ha utilizado en una amplia variedad de aplicaciones criptográficas y se utiliza comúnmente para verificar la integridad de los datos en las investigaciones forenses. Este algoritmo fue diseñado por Ronald Rivest en 1991 y ha sido muy utilizado desde entonces.

Una gran ventaja de MD5 es que calcula más rápido y produce pequeños hashes. Los pequeños hashes son un importante punto de interés cuando es necesario almacenar miles de estos hashes en una investigación forense. Imagínese cuántos archivos tendrá una PC común en su disco duro. Si necesita calcular un hash de cada uno de estos archivos y almacenarlos en una base de datos, haría una gran diferencia si cada hash calculado tiene 16 bytes o 32 bytes de tamaño.

Hoy en día, la principal desventaja de MD5 es el hecho de que ya no se considera que sea resistente a las colisiones. Esto significa que es posible calcular el mismo hash de dos entradas diferentes. Teniendo esto en cuenta, ya no es posible garantizar que un archivo no ha sido modificado simplemente comparando su hash MD5 en dos etapas diferentes de una investigación. En este momento es posible crear una colisión muy rápido, (ver http://www.win.tue.nl/hashclash/On%20Collisions%20for%20MD5%20-%20 MMJ% 20Stevens.pdf) pero sí sigue siendo difícil de modificar un archivo de una manera, ahora que es una versión maliciosa de ese archivo benigno, y mantener el hash MD5 del archivo original.

El muy famoso criptógrafo, Bruce Schneier, escribió una vez (`https://www.schneier.com/blog/archives/2008/12/forging_ssl_cer.html`):

> *"Ya sabíamos que MD5 es una función de hash roto "y que" nadie debería estar usando MD5 más".*

No vamos a ir tan lejos (sobre todo porque un montón de herramientas y servicios todavía utilizan MD5), pero usted debe tratar de cambiar a SHA256 o al menos comprobar sus resultados con la ayuda de diferentes funciones hash en los casos en que es crítico. Siempre que la cadena de custodia sea crucial, recomendamos usar algoritmos de hash múltiples para probar la integridad de sus datos.

SHA256

SHA-2 es un conjunto de funciones de hash criptográficas diseñadas por la NSA (Agencia Nacional de Seguridad de los Estados Unidos) y significa "Secure Hash Algorithm 2nd Generation". Ha sido publicado en 2001 por el NIST como una norma federal estadounidense (FIPS). La familia SHA-2 consta de varias funciones hash con digest (valores de hash) que están entre 224 bits y 512 bits. Las funciones criptográficas SHA256 y SHA512 son las versiones más comunes de las funciones hash SHA-2 calculadas con palabras de 32 bits y 64 bits.

A pesar de que estos algoritmos calculan más lento y que los hashes calculados son más grandes en tamaño (en comparación con MD5), deben ser los algoritmos preferidos que se utilizan para las verificaciones de integridad durante las investigaciones forenses. Hoy en día, SHA256 es una función de hash criptográfica ampliamente utilizada que sigue siendo resistente a las colisiones y totalmente fiable.

SSDEEP

La diferencia más grande entre **MD5, SHA256 y SSDEEP** es el hecho de que **SSDEEP** no se considera una **función de hash criptográfica**, ya que sólo cambia ligeramente cuando la entrada se cambia por un bit. Por ejemplo:

```
user@lab:~$ echo -n This is a test message | ssdeep
ssdeep,1.1--blocksize:hash:hash,filename
```

```
3:hMCEpFzA:hurs,"stdin"

user@lab:~$ echo -n This is A test message | ssdeep
ssdeep,1.1--blocksize:hash:hash,filename
3:hMCkrzA:hOrs,"stdin"
```

Este comportamiento no es una debilidad de SSDEEP, es una gran ventaja de esta función. En realidad, SSDEEP es un programa para calcular y hacer coincidir los valores de **Context Triggered Piecewise Hashing (CTPH)**. CTPH es una técnica que también se conoce como Hashing Fuzzy y es capaz de coincidir con las entradas que tienen homologías. Entradas con homologías tienen secuencias de bytes idénticos en un orden dado con bytes totalmente diferentes entre ellos. Estos bytes entre ellos pueden diferir en contenido y longitud. CTPH, originalmente basado en el trabajo del Dr. Andrew Tridgell, fue adaptado por Jesse Kornblum y publicado en la conferencia de DFRWS en 2006 en un papel llamado Identificación de Archivos Casi Idénticos Usando **Context Triggered Piecewise Hashing**; consulte http://dfrws.org/ 2006/proceedings/12-Kornblum.pdf.

SSDEEP se puede usar para comprobar la similitud de los dos archivos y en qué parte se encuentra la diferencia. Esta característica se utiliza a menudo para comprobar si dos aplicaciones diferentes en los dispositivos móviles tienen una base de código común, como se muestra a continuación:

```
user@lab:~$ ssdeep -b malware-sample01.apk > signature.txt

user@lab:~$ cat signature.txt
Ssdeep,1.1--blocksize:hash:hash,filename
49152:FTqSf4xGvFowvJxThCwSoVpzPb03++4zlpBFrnInZWk:JqSU4ldVVpDIcz3BFr8Z7,
" malware-sample01.apk"

user@lab:~$ ssdeep –mb signature.txt malware-sample02.apk
malware-sample02.apk matches malware-sample01.apk (75)
```

En el ejemplo anterior, puede ver que la segunda muestra coincide con la primera con una probabilidad muy alta. Estos coincidencias indican la potencial reutilización del código fuente o al menos un gran número de archivos dentro del archivo apk son idénticos. Un examen manual de los archivos en cuestión se requiere para decir exactamente qué partes del código o archivos son idénticos; sin embargo, ahora sabemos que ambos archivos son similares entre sí.

Apoyo a la cadena de custodia

Los resultados de las investigaciones forenses pueden tener un impacto severo en las organizaciones y los individuos. Dependiendo de su campo de trabajo, su investigación puede convertirse en evidencia en el tribunal.

En consecuencia, la integridad de la evidencia forense debe garantizarse no sólo en la recopilación de las pruebas, sino también durante toda la manipulación y el análisis. Por lo general, el primer paso en una investigación forense es reunir las pruebas. Normalmente, esto se hace usando una copia bit a bit de los medios originales. Todo el análisis posterior se realiza en esta copia forense.

Creación de sumas de hash de imágenes de disco completas

Para asegurarse de que una copia forense es realmente idéntica a los medios originales, las sumas hash de los medios y de la copia forense se hacen. Estas sumas hash deben coincidir para probar que la copia es exactamente como los datos originales. Hoy en día, se ha vuelto común usar al menos dos algoritmos de hash criptográficos diferentes para minimizar el riesgo de colisiones hash y endurecer el proceso general contra ataques de colisión hash.

Con Linux, uno puede crear fácilmente hashes MD5 y SHA256 desde una unidad o varios archivos. En el ejemplo siguiente, calcularemos las sumas MD5 y las sumas SHA256 para dos archivos, para proporcionar una prueba de contenido idéntico:

```
user@lab:~$ md5sum /path/to/originalfile
/path/to/forensic_copy_of_sdb. img


user@lab:~$ sha256sum /path/to/originalfile
/path/to/forensic_copy_of_ sdb.img
```

Esta prueba de idéntico contenido se requiere para apoyar la cadena de custodia, es decir, para demostrar que los datos analizados son idénticos a los datos en bruto en el disco. El término **sdb** se refiere a una unidad conectada a la estación de trabajo forense (en Linux, el **segundo disco duro** se llama **sdb**). Para apoyar aún más la cadena de custodia, se recomienda utilizar un dispositivo de bloqueo de escritura entre la evidencia y la estación de trabajo forense para evitar cualquier cambio accidental de la evidencia. El segundo argumento representa la ubicación de una copia bit a bit de la evidencia. Los comandos emiten las sumas hash para la unidad original y la copia. La copia puede considerarse forense si ambas sumas MD5 coinciden y ambas sumas SHA256 coinciden.

Si bien el método mostrado en el ejemplo anterior funciona, tiene una gran desventaja, la evidencia y su copia tienen que leerse dos veces para calcular las sumas de hash. Si el disco es un disco duro de 1 TB, puede ralentizar el proceso en varias horas.

El siguiente código Python lee los datos una sola vez y lo inyecta a dos cálculos de hash. Por lo tanto, este script de Python es casi el doble de rápido que ejecutar md5sum seguido de sha256sum y produce exactamente las mismas sumas hash que estas herramientas:

```
#!/usr/bin/env python

import hashlib
import sys

def multi_hash(filename):
    """Calculates the md5 and sha256 hashes
       of the specified file and returns a list
       containing the hash sums as hex strings."""

    md5 = hashlib.md5()
    sha256 = hashlib.sha256()

    with open(filename, 'rb') as f:
        while True:
            buf = f.read(2**20)
            if not buf:
                break
            md5.update(buf)
            sha256.update(buf)

    return [md5.hexdigest(), sha256.hexdigest()]


if __name__ == '__main__':
    hashes = []
    print '---------- MD5 sums ----------'
    for filename in sys.argv[1:]:
        h = multi_hash(filename)
        hashes.append(h)
        print '%s  %s' % (h[0], filename)

    print '---------- SHA256 sums ----------'
    for i in range(len(hashes)):
        print '%s  %s' % (hashes[i][1], sys.argv[i+1])
```

En la siguiente llamada de script, calculamos las sumas hash de algunas de las herramientas comunes de Linux:

```
user@lab:~$ python multihash.py /bin/{bash,ls,sh}
---------- MD5 sums ----------
d79a947d06958e7826d15a5c78bfaa05  /bin/bash
fa97c59cc414e42d4e0e853ddf5b4745  /bin/ls
c01bc66da867d3e840814ec96a137aef  /bin/sh
---------- SHA256 sums ----------
cdbcb2ef76ae464ed0b22be346977355c650c5ccf61fef638308b8da60780bdd  /bin/
bash
846ac0d6c40d942300de825dbb5d517130d8a0803d22115561dcd85efee9c26b  /bin/ls
e9a7e1fd86f5aadc23c459cb05067f49cd43038f06da0c1d9f67fbcd627d622c  /bin/sh
```

Es crucial documentar las sumas hash de los datos originales y la copia forense en el informe forense. Un partido independiente puede entonces leer la misma pieza de evidencia y confirmar que los datos que usted analizó son exactamente los datos de la evidencia.

Creación de sumas de hash de árboles de directorios

Una vez copiada la imagen completa, su contenido debe ser indexado y el hashdeben crear sumas para cada archivo. Con el soporte de la función `multi_hash` previamente definida y las bibliotecas estándar de Python, se puede crear una plantilla de informe que contenga una lista de todos los nombres de archivo, tamaños y valores de hash, como se muestra a continuación:

```
#!/usr/bin/env python

from datetime import datetime
import os
from os.path import join, getsize
import sys
from multihash import multi_hash

def dir_report(base_path, reportfilename):
    """Creates a report containing file integrity information.

    base_path -- The directory with the files to index
    reportfilename -- The file to write the output to"""

    with open(reportfilename, 'w') as out:
        out.write("File integrity information\n\n")
        out.write("Base path:      %s\n" % base_path)
```

```
        out.write("Report created: %s\n\n" % datetime.now().
isoformat())
        out.write('"SHA-256","MD5","FileName","FileSize"')
        out.write("\n")

        for root, dirs, files in os.walk(base_path):
            write_dir_stats(out, root, files)

        out.write("\n\n--- END OF REPORT ---\n")

def write_dir_stats(out, directory, files):
    """Writes status information on all specified files to the report.

    out -- open file handle of the report file
    directory -- the currently analyzed directory
    files -- list of files in that directory"""

    for name in files:
        fullname = join(directory, name)
        hashes = multi_hash(fullname)
        size = getsize(fullname)
        out.write('"%s","%s","%s",%d' % (hashes[1], hashes[0],
fullname, size))
        out.write("\n")


if __name__ == '__main__':
    if len(sys.argv) < 3:
        print "Usage: %s reportfile basepath\n" % sys.argv[0]
        sys.exit(1)

    dir_report(sys.argv[2], sys.argv[1])
```

Este script de Python es todo lo que se necesita para generar la información de integridad de un árbol de directorios que incluye tamaños de archivo, nombres de archivo y sumas de hash (SHA256, MD5). A continuación se muestra una llamada de ejemplo en nuestro directorio de secuencias de comandos:

```
user@lab:/home/user/dirhash $ python dirhash.py report.txt
. user@lab:/home/user/dirhash $ cat report.txt
File integrity information


Base path:      .
```

```
Report created: 2015-08-23T21:50:45.460940

"SHA-256","MD5","FileName","FileSize"
"a14f7e644d76e2e232e94fd720d35e59707a2543f01af4123abc46e8c10330cd","9c0d1f70fffe5c59a7700b2b9bfd50cc","./multihash.py",879
"a4168e4cc7f8db611b339f4f8a949fbb57ad893f02b9a65759c793d2c8b9b4aa","bcf5a41a403bb45974dd0ee331b1a0aa","./dirhash.py",1494
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855","d41d8cd98f00b204e9800998ecf8427e","./report.txt",0
"03047d8a202b03dfc5a310a81fd8358f37c8ba97e2fff8a0e7822cf7f36b5c83","416699861031e0b0d7b6d24b3de946ef","./multihash.pyc",1131


--- END OF REPORT ---
```

Sin embargo, el archivo de informe resultante en sí no tiene ninguna protección de integridad. Se recomienda firmar el informe resultante, por ejemplo, utilizando **GnuPG**, como se muestra a continuación:

```
user@lab:~$ gpg --clearsign report.txt
```

Si nunca ha utilizado gpg antes, necesita generar una clave privada antes de poder firmar los documentos. Esto se puede hacer con el comando `gpg --gen-key`. Consulte https://www.gnupg.org/documentation para obtener más detalles sobre GnuPG y su uso. Esto crea un archivo adicional report.txt.asc que contiene el informe original y una firma digital. Cualquier modificación posterior de ese archivo invalida la firma digital.



Las técnicas descritas aquí son meramente los ejemplos de cómo apoyar la cadena de custodia. Si el análisis forense debe ser usado en la corte, se recomienda altamente buscar consejo legal sobre los requisitos de la cadena de custodia en su legislación.

Escenarios del mundo real

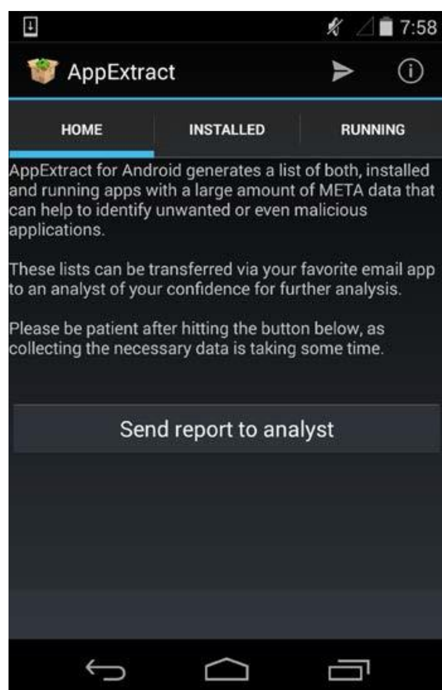
Esta sección demostrará algunos casos de uso en los que los algoritmos y técnicas anteriores se utilizan para apoyar al investigador. Para este capítulo, usamos dos ejemplos muy comunes e interesantes, **Malware móvil** y la **Biblioteca Nacional de Referencia de Software (NSRL).**

Malware móvil

En este ejemplo, comprobaremos las aplicaciones instaladas en un smartphone Android en un sistema de análisis en línea, **Mobile-Sandbox**. Mobile-Sandbox es un sitio web que ofrece archivos gratuitos de Android para detectar virus o comportamiento sospechoso, http://www.mobilesandbox.org. Está conectado a VirusTotal, que utiliza hasta 56 diferentes productos de antivirus y motores de análisis para detectar virus que la solución de antivirus del usuario puede haber no detectado o verificar en contra de cualquier falso positivo. Además, Mobile-Sandbox utiliza técnicas personalizadas para detectar aplicaciones que actúan potencialmente malintencionadas. Proveedores de software de antivirus, desarrolladores e investigadores detrás de Mobile-Sandbox pueden recibir copias de los archivos para ayudar en la mejora de su software y técnicas.

En el ejemplo, usaremos dos pasos para comparar correctamente las aplicaciones instaladas con las aplicaciones ya probadas en el servicio web de Mobile-Sandbox.

El primer paso es obtener las sumas de hash de las aplicaciones instaladas en el dispositivo. Esto es muy importante ya que estos valores pueden ayudar a identificar las aplicaciones y comprobarlas con los servicios en línea. Para este ejemplo, utilizaremos una aplicación de Google Play, **AppExtract** (https://play.google.com/store/apps/details?id=de.mspreitz. Appextract). La manera forense más correcta de obtener estos valores se puede encontrar en el Capítulo 6, utilizar Python para forense móvil.



AppExtract para Android genera una lista de aplicaciones instaladas y en ejecución con una gran cantidad de metadatos que pueden ayudar a identificar aplicaciones no deseadas o incluso maliciosas. Estos metadatos contienen la suma hash SHA256 de los paquetes de aplicaciones, un indicador de si la aplicación ha sido instalada por el usuario o el propio sistema y una gran cantidad de datos adicionales que pueden ayudar a identificar si la aplicación es benigna o no. Estas listas se pueden transferir a través de su aplicación de correo electrónico favorita para un análisis más detallado. Una vez que reciba el correo electrónico de texto sin formato con las listas generadas, sólo tendrá que copiar la lista que contiene todas las aplicaciones instaladas en un archivo CSV. Este archivo se puede usar para un análisis automatizado o abrirlo con **LibreOffice Calc** en el entorno de laboratorio. Puede ver los metadatos de la versión actual del navegador Chrome para Android en los siguientes apartados:

```
Type;App_Name;md5;TargetSdkVersion;Package_Name;Process_Name;APK_
Location;Version_Code;Version_Name;Certificate_Info;Certificate_
SN;InstallTime;LastModified


SystemApp;Chrome;4e4c56a8a7d8d6b1ec3e0149b3918656;21;com.android.
chrome;com.android.chrome;/data/app/com.android.chrome-
2.apk;2311109; 42.0.2311.109;CN=Android, OU=Android, O=Google Inc.,
L=Mountain View, ST=California,
C=US;14042372374541250701;unknown;unknown
```

El segundo paso es comparar las sumas hash del dispositivo (tercera columna en nuestro archivo CSV) con la base de datos Mobile-Sandbox. Esto se puede hacer con la ayuda de la siguiente secuencia de comandos que guardaremos como get_infos_mobilesandbox.py:

```
#!/usr/bin/env python

import sys, requests

# Authentication Parameters
# if you need an API key and user name please contact @m_spreitz
API_FORMAT = 'json'
API_USER = ''
API_KEY = ''
# parsing input parameters
if (len(sys.argv) < 3):


    print "Get infos to a specific Android app from the Mobile-
Sandbox."
    print "Usage: %s requests [type (md5,sha256)] [value]" % sys.
argv[0]
    sys.exit(0)

# building the payload
payload = {'format':API_FORMAT,
```

```
            'username':API_USER,
            'api_key':API_KEY,
            'searchType':str(sys.argv[1]),   # has to be md5 or sha256
            'searchValue':str(sys.argv[2])}

# submitting sample hash and getting meta data
print "--------------------------------"
r = requests.get("http://mobilesandbox.org/api/bot/queue/get_info/",
params=payload)

# printing result and writing report file to disk
if not r.status_code == requests.codes.ok:
    print "query result: \033[91m" + r.text + "\033[0m"
else:
    for key, value in r.json().iteritems():
        print key + ": \033[94m" + str(value) + "\033[0m"
print "--------------------------------"
```

El script se puede utilizar como se muestra:

```
(labenv)user@lab:~$ ./get_infos_mobilesandbox.py md5
4e4c56a8a7d8d6b1ec3e0149b3918656


--------------------------------
status: done
min_sdk_version: 0
package_name: com.android.chrome
apk_name: Chrome.apk
AV_detection_rate: 0 / 56
drebin_score: benign (1.38173)
sample_origin: user upload
android_build_version: Android 1.0
ssdeep:
196608:ddkkKqfC+ca8eE/jXQewwn5ux1aDn9PpvPBic6aQmAHQXPOo:dBKZaJYXQ
E5u3ajtpvpeaQm1
sha256: 79de1dc6af66e6830960d6f991cc3e416fd3ce63fb786db6954a3ccaa7f7323c
malware_family: ---
md5: 4e4c56a8a7d8d6b1ec3e0149b3918656
--------------------------------
```

Con la ayuda de estas tres herramientas, es posible comprobar rápidamente si una aplicación en un dispositivo móvil está potencialmente infectada (ver las partes resaltadas en la respuesta) o al menos por dónde comenzar con la investigación manual si una aplicación no ha sido probada antes.

NSRLquery

Para aumentar la eficiencia en el análisis forense, es crucial ordenar los archivos que pertenecen a software conocido y no han sido modificados. La **National Software Reference Library (NSRL)** mantiene múltiples listas de sumas de hash para el contenido conocido. NSRL es un proyecto del Departamento de Seguridad Nacional de los Estados Unidos, más detalles están disponibles en http://www.nsrl.nist.gov/. Es importante entender que estas listas de sumas de hash simplemente indican que un archivo no fue modificado en comparación con la versión que fue presentada al NSRL. En consecuencia, es normal que una gran cantidad de archivos, que deben ser analizados durante una investigación forense, no están listados en NSRL. Por otro lado, incluso los archivos listados pueden ser utilizados e implementados por un atacante como una herramienta. Por ejemplo, una herramienta como psexec.exe es un programa proporcionado por Microsoft para administración remota y que aparece en NSRL. Sin embargo, un atacante puede haberlo desplegado para sus fines maliciosos.



¿Qué lista de NSRL se debe usar?

NSRL consta de varios conjuntos de hash. Se recomienda encarecidamente comenzar con el *minimal set*. Este conjunto solo contiene una suma de hash por archivo, lo que significa que solo se conoce una versión de archivo.

El conjunto mínimo se ofrece de forma gratuita para descargar en la página principal del NIST. La descarga consta de un único archivo ZIP con la lista hash y una lista de productos de software compatibles como los contenidos más destacados.

Los hashes se almacenan en el archivo `NSRLFile.txt` que contiene un hash de archivo por línea, por ejemplo:

```
"3CACD2048DB88F4F2E863B6DE3B1FD197922B3F2","0BEA3F79A36B1F67B2CE0F595
5 24C77C","C39B9F35","TWAIN.DLL",94784,14965,"358",""
```

Los campos de este registro son los siguientes:

- La suma de hash del archivo que se calcula con SHA-1, un predecesor del algoritmo SHA-256 descrito anteriormente.
- La suma de hash del archivo que se calcula con MD5.
- La suma de comprobación CRC32 del archivo.
- El nombre del archivo.

- El tamaño del archivo en bytes.
- Un código de producto que indica el producto de software al que pertenece este archivo. El archivo `NSRLProd.txt` contiene una lista de todos los productos y puede utilizarse para buscar el código del producto. En el ejemplo anterior, el código 14965 denota Microsoft Picture It !.
- El sistema operativo donde se espera este archivo. La lista de códigos del sistema operativo se puede encontrar en `NSRLOS.txt`.
- Un indicador de si este archivo debe considerarse normal (""), un archivo malicioso ("N") o especial ("S"). Mientras este indicador forma parte de la especificación, todos los archivos del conjunto mínimo NSRL actual se establecen como normales.

Más detalles sobre las especificaciones de archivo se pueden encontrar en `http://www.nsrl.nist.gov/Documents/Data-Formats-of-the-NSRL-Reference-Data-Set-16.pdf`.

Descarga e instalación de nsrlsvr

Actualmente, la base de datos NSRL contiene más de 40 millones de hashes distintos en un conjunto mínimo. Una búsqueda basada en texto tardaría unos minutos, incluso en una estación de trabajo actualizada. Por lo tanto, es importante realizar búsquedas eficaces en esa base de datos. La herramienta **nsrlsvr** de Rob Hanson proporciona un servidor que soporta búsquedas eficientes. Está disponible en https://rjhansen.github.io/nsrlsvr/.



También hay servidores públicos NSRL en Internet que puede utilizar. Estos son generalmente realizadas sobre una base como está. Sin embargo, para probar conjuntos más pequeños de hashes, puede utilizar el servidor público de Robert Hanson **nsrllookup.com** y continuar leyendo la siguiente sección.

Para compilar el software en un sistema Linux, las herramientas de compilador c ++, automake y autoconf deben estar instaladas. Las instrucciones de instalación detalladas, incluyendo todos los requisitos, se proporcionan en el archivo `INSTALL`.



Instalación de nsrlsvr en un directorio que no sea el predeterminado

El directorio de instalación de nsrlsvr se puede cambiar llamando al script de `configuración` con el parámetro `--prefix`. El valor del parámetro indica el directorio de destino. Si se especifica un directorio que puede ser escrito por el usuario, la instalación no requiere privilegios de root y puede quitarse por completo eliminando el directorio de instalación.

El nsrlsrv mantiene su propia copia de todas las sumas hash MD5 de la base de datos NSRL. Por lo tanto, es necesario inicializar la base de datos hash. La herramienta nsrlupdate necesaria se proporciona con nsrlsrv.

```
user@lab:~$ nsrlupdate your/path/to/NSRLFile.txt
```

Después de que la base de datos esté completamente poblada, el servidor se puede

iniciar simplemente llamando a:

```
user@lab:~$ nsrlsvr
```

Si todo se instala correctamente, este comando se devuelve sin proporcionar ninguna salida y el servidor comienza a escuchar el puerto TCP 9120 para las solicitudes.

Escribir un cliente para nsrlsvr en Python

También hay una herramienta de cliente para usar nsrlsvr llamada **nsrllookup**. El cliente está escrito en C ++ y esta disponible en https://rjhansen.github.io/nsrllookup/. Sin embargo, un cliente para interactuar con nsrlsvr puede ser fácilmente implementado en Python nativo. En esta sección se explica el protocolo y se muestra una implementación un ejemplo de dicho cliente.
El nsrlsvr implementa un protocolo orientado a texto en su puerto de red 9120. Cada comando consta de una línea de texto seguida de una nueva línea (CR LF). Se admiten los siguientes comandos:

- version: 2.0: El comando version se utiliza para el handshake inicial entre el cliente nsrl y nsrlsvr. Se supone que el cliente debe proporcionar su versión después del colon. El servidor siempre responderá con OK seguido de un salto de línea.
- **query 5CB360EF546633691912089DB24A82EE 908A54EB629F410C647A573F91E80775 BFDD76C4DD6F8C0C2474215AD5E193CF**: El comando de consulta se utiliza para consultar realmente la base de datos NSRL desde el servidor. La consulta de **palabras** clave es seguida por una o varias sumas de hash MD5. El servidor responderá con OK seguido de una secuencia de ceros y unos. Un 1 indica que la suma hash MD5 se encontró en la base de datos y un 0 indica que no había coincidencia. Por ejemplo, la consulta mostrada anteriormente llevaría a la siguiente respuesta:

  ```
  OK 101
  ```

  Esto significa que el primer y el último hash MD5 se encontraron en NSRL, pero no se pudo encontrar la suma de hash media.
- **BYE**: El comando bye termina la conexión con el nsrlsvr.

En consecuencia, la siguiente rutina de Python es suficiente para consultar eficientemente la base de datos NSRL:

```
#!/usr/bin/env python

import socket

NSRL_SERVER='127.0.0.1'
NSRL_PORT=9120

def nsrlquery(md5hashes):
    """Query the NSRL server and return a list of booleans.

    Arguments:
    md5hashes -- The list of MD5 hashes for the query.
    """

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((NSRL_SERVER, NSRL_PORT))

    try:
        f = s.makefile('r')
        s.sendall("version: 2.0\r\n")
        response = f.readline();
        if response.strip() != 'OK':
            raise RuntimeError('NSRL handshake error')

        query = 'query ' + ' '.join(md5hashes) + "\r\n"
        s.sendall(query)
        response = f.readline();

        if response[:2] != 'OK':
            raise RuntimeError('NSRL query error')

        return [c=='1' for c in response[3:].strip()]
    finally:
        s.close()
```

El uso de este módulo es tan fácil como se muestra aquí:

```
import nsrlquery
hashes = ['86d3d86902b09d963afc08ea0002a746',
          '3dcfe9688ca733a76f82d03d7ef4a21f',
          '976fe1fe512945e390ba10f6964565bf']
nsrlquery.nsrlquery(hashes)
```

Este código consulta el servidor NSRL y devuelve una lista de booleanos, cada uno indicando si el hash MD5 correspondiente se ha encontrado en la lista de archivos NSRL.

Resumen

Este capítulo proporciona una visión general de los dominios de los algoritmos forenses y de ejemplo para cada uno de estos dominios. También le mostramos cómo comparar aplicaciones instaladas en un dispositivo Android con servicios web como **Mobile-Sandbox**. En el segundo ejemplo del mundo real, hemos demostrado cómo ordenar los archivos benignos y conocidos de un sistema Windows para reducir la cantidad de datos que se van a analizar manualmente. Con **NSRLquery**, las investigaciones forenses pueden centrarse en contenido nuevo o modificado y no necesitan perder tiempo en el contenido ampliamente conocido de las aplicaciones estándar.

En los siguientes capítulos, estos algoritmos se aplicarán a una selección de tipos de dispositivos, sistemas operativos y aplicaciones para su uso durante la investigación forense.

3

Uso de Python para Windows y Linux Forensics

En este capítulo, nos centraremos en las partes de la investigación forense que son específicas de los sistemas operativos. Elegimos los sistemas operativos más utilizados en los sistemas de escritorio y servidores: Microsoft Windows y Linux.

Para ambos sistemas operativos, seleccionamos ejemplos de evidencia interesante y cómo automatizar su análisis usando Python. En consecuencia, en este capítulo aprenderá lo siguiente:

- Analizar los fundamentos del registro de eventos de Windows, seleccionar partes interesantes y analizarlos automáticamente
- Organización del Registro de Windows y búsqueda eficiente de **Indicadores de Compromiso (IOC)**
- Buscar información de la cuenta local de Linux para COI
- Entender, usar y analizar los metadatos de archivos Linux con POSIX ACL y capacidades basadas en archivos como las extensiones más destacadas de los metadatos estándar

Analizando el registro de eventos de Windows

Windows incluye muchas capacidades de monitoreo y registro y rastrea los datosy eventos para una gran cantidad y variedad de actividades que ocurren en el sistema operativo. El gran número de eventos, que se pueden registrar, tampoco facilita que un administrador identifique los eventos importantes específicos ni ayuda a un investigador forense a encontrar Indicadores de Compromiso. Por lo tanto, comenzaremos esta sección con una pequeña introducción al registro de eventos de Windows y los cambios en su formato en el tiempo, seguido de una descripción de los tipos de eventos importantes que deberían ayudar a un investigador a encontrar rápidamente acciones sospechosas en la gran cantidad de otros eventos. En la última sección de este capítulo, demostraremos cómo analizar el registro de sucesos y encontrar automáticamente las posibles IOCs (por ejemplo, inicios de sesión de usuarios, creación de servicios, etc.).

El registro de eventos de Windows

Según Microsoft, los archivos del registro de eventos de Windows son archivos especiales que registran eventos significativos, como cuando un usuario inicia sesión en el equipo o cuando un programa encuentra un error (consulte http://windows.microsoft.com/en-us/windows/ what-information-event-logs-event-viewer#1TC=windows-7). Cuando se producen estos tipos de eventos, Windows registra el evento en un registro de eventos que se puede leer mediante el Visor de eventos o herramientas similares.

Con la publicación de Windows 7 y Windows Server 2008, Microsoft ha realizado un cambio importante en su técnica de registro de eventos. Cambiaron del clásico **registro de eventos de Windows (EVT)** al nuevo **registro de eventos de Windows XML (EVTX)**. En los párrafos siguientes, explicaremos algunas de las principales diferencias entre estos dos tipos de archivo de registro.

Debido al hecho de que Microsoft ya no soporta Windows XP y Server 2003 está en la fase de soporte extendido en el presente (lo que significa que saldrá de soporte muy pronto), hay sistemas XP y 2003 todavía por ahí. Así, algunos investigadores todavía van a necesitar saber la diferencia entre el EVT más viejo y el nuevo EVTX y los problemas posibles que surgen durante el análisis de estos archivos.

Además de las diferencias binarias en los registros y los archivos de registro de eventos en si mismos, la cantidad de estos archivos de registro también difiere. En un sistema Windows XP / 2003, había tres archivos de registro de eventos principales: **Sistema, Aplicación y Seguridad**. Se almacenan en el directorio C:\Windows\system32\config. Las versiones de servidor del sistema operativo pueden mantener registros de eventos adicionales (servidor DNS, servicio de directorio, servicio de replicación de archivos, etc.) según la funcionalidad del servidor. En un sistema Windows 7 actual, puede encontrar más de 143 archivos llenos de registros de eventos. Esto obtiene aún más si lo comparas con las versiones de servidor más nuevas de Microsoft Windows

Los registros de registro EVT sólo contienen una cantidad muy pequeña de contenido legible por humanos y se hacen legibles por medio de herramientas tales como el visor de eventos en el momento del análisis. Estas herramientas combinan las plantillas de registro predefinidas que se almacenan comúnmente en los archivos DLL o EXE del sistema con los datos almacenados en el propio archivo EVT. Cuando una de las diversas herramientas de visualización de registros muestra registros de registro, tiene que determinar qué archivos DLL almacenarán las plantillas de mensaje. Esta meta-información se almacena en el Registro de Windows y es específica para cada tipo de los tres archivos de registro de sucesos (Sistema, Aplicación y Seguridad) mencionados anteriormente.

Todos los detalles mencionados anteriormente siguen el hecho de que los archivos EVT no son realmente útiles sin sus metarchivos correspondientes, que almacenan el significado central del registro. Esto crea dos problemas de análisis principales:

- En primer lugar, un atacante podría modificar archivos DLL o el Registro de Windows para cambiar el significado de los registros de eventos sin tener que tocar el archivo EVT.
- En segundo lugar, cuando el software se desinstala en un sistema, podría hacer que los registros EVT perdieran su contexto.

Como investigador, hay que tener cuidadosamente en cuenta estas cuestiones cuando se analizan registros EVT y también cuando se escriben esos registros en sistemas remotos para análisis posterior. Un análisis aún más detallado de los registros EVT se puede encontrar en el ForensicsWiki, http://forensicswiki.org/wiki/Windows_Event_Log_(EVT).

En comparación con EVT, los archivos EVTX se almacenan como un formato de archivo XML binario. En los sistemas Windows más recientes, los registros de eventos se pueden ver y analizar con el Visor de eventos o con un gran número de otros programas y herramientas (en las siguientes secciones, describiremos algunos scripts de Python que se pueden utilizar también). Cuando se utiliza el visor de eventos, hay que tener en cuenta que este programa puede representar los archivos EVTX en dos formatos diferentes: **general** y **detallada**. La vista general (a veces llamada formateada) puede ocultar datos de eventos significativos que se almacenan en el registro de eventos y sólo se pueden ver en la vista detallada. Por lo tanto, si está planeando utilizar el Visor de eventos para analizar los archivos EVTX, utilice siempre la opción detallada para mostrar los archivos.

Si estás interesado en un análisis más detallado del formato de archivo EVTX, deberías echar un vistazo a ForensicsWiki, http://forensicswiki.org/wiki/Windows_ XML_Event_Log_ (EVTX). Otra gran explicación de los detalles más profundos del formato de archivo EVTX ha sido presentada por Andreas Schuster en DFRWS 2007, consulte http://www.dfrws.org/2007/proceedings/p65-schuster_pres.pdf. Esta presentación puede ser muy útil si desea comprender los detalles del formato XML binario o escribir sus propios analizadores de archivos EVTX.

Si necesita abrir los archivos EVT en un sistema Windows 7 o superior, es mejor convertir el archivo EVT antiguo a la sintaxis EVTX antes de abrirlo. Esto se puede hacer de varias maneras como se describe en una entrada del blog http://blogs.technet. com/b/askperf/archive/2007/10/12/windows-vista-and-exported-event-log-files.aspx.

Eventos interesantes

Una lista completa de eventos de Windows en el sistema más reciente se puede encontrar en un artículo de la base de conocimiento de Microsoft en, https://support.microsoft.com/en-us/kb/947226. A medida que el número de estos eventos es cada vez mayor con cada nueva versión del sistema y cada aplicación recién instalada, puede encontrar fácilmente más de varios cientos de tipos de eventos diferentes en un único sistema Windows. Debido a este hecho, hemos tratado de resolver algunos tipos de eventos interesantes que pueden ser útiles al analizar un sistema o reconstruir eventos de usuario (una explicación más detallada de qué registros de eventos pueden ser útiles, en qué condiciones también se pueden encontrar en TSA-13- 1004-SG, https://www. nsa.gov/ia/_files/app/spotting_the_adversary_with_windows_event_log_ monitoring.pdf):

- Si la organización está utilizando activamente el Microsoft **Enhanced Mitigation Experience Toolkit (EMET)**, estos registros pueden ser muy útiles durante la investigación.
- **Windows-Update-Failure (20, 24, 25, 31, 34, 35)**: El problema de no solucionar los problemas debe ser tratado para evitar prolongar la existencia de un problema de aplicación o vulnerabilidad en el sistema operativo o una aplicación. A veces, esto también ayuda en la identificación de infecciones de un sistema.
- **Microsoft-Windows-Eventlog (104, 1102)**: Es poco probable que los datos del registro de eventos se borren durante las operaciones normales y es más probable que un atacante malicioso pueda tratar de cubrir sus pistas borrando un registro de eventos. Cuando se borra un registro de eventos, es sospechoso.
- **Microsoft-Windows-TaskScheduler (106)**: Muestra Tareas programadas recién registradas. Esto puede ser muy útil si está buscando signos de infecciones de malware.
- **McAfee-Log-Event (257)**: McAfee AntiVirus puede detectar comportamientos de malware sin detectar realmente el archivo EXE. Esto puede ser muy valioso en la determinación de cómo el malware entró en un sistema. En general, los registros de eventos de la solución AV instalada son registros muy valiosos al iniciar un análisis de un sistema potencialmente comprometido. Por lo tanto, debe recordar dónde encontrar esos registros en el registro de eventos.

- **Microsoft-Windows-DNS-Client (1014)**: Tiempo de espera de resolución de nombres DNS; este tipo de evento también puede ser muy útil cuando se busca malware o cuando se intenta averiguar si un usuario ha intentado conectarse a un sitio web o servicio específico.
- **Firewall-Rule-Add/Change/Delete (2004, 2005, 2006, 2033)**: Si las estaciones de trabajo cliente están aprovechando el Firewall de Windows basado en host incorporado, entonces hay valor en la recopilación de eventos para rastrear el estado del cortafuegos. Los usuarios normales no deben estar modificando las reglas de firewall de su máquina local.
- **Microsoft-Windows-Windows Defender (3004)**: Registros de detección de malware de Windows Defender.
- **Microsoft-Windows-Security-Auditing (4720, 4724, 4725, 4728, 4732, 4635, 4740, 4748, 4756)**: En estos registros, puede encontrar información como los inicios de sesión de escritorio remoto y los usuarios que se han agregado a grupos privilegiados y también se pueden rastrear los bloqueos de cuentas. Las cuentas de usuario que se están promoviendo a los grupos privilegiados deben ser auditadas muy de cerca para asegurarse de que, de hecho, se supone que los usuarios están en un grupo privilegiado. La membresía no autorizada de los grupos privilegiados es un fuerte indicador de que se ha producido una actividad maliciosa.
- **Service-Control-Manager (7030, 7045)**: Supervisa si un servicio está configurado para interactuar con el escritorio o se ha instalado en el sistema en general.
- **App-Locker-Block/Warning (8003, 8004, 8006, 8007)**: Los eventos de listas blancas de aplicaciones deben recopilarse para buscar las aplicaciones que han sido bloqueadas desde la ejecución. Cualquier aplicación bloqueada podría ser malware o los usuarios que intenten ejecutar un software no aprobado.

Harlan Carvey declaró en uno de sus posts en el blog ((http://windowsir.blogspot.de/2014/10/windows-event-logs.html) que más allá de los registros de eventos individuales (source/ID pairs), uno de los aspectos de las versiones más recientes de Windows (en particular, Windows 7) es que hay un montón de eventos que se están grabando de forma predeterminada en varios archivos de registro de eventos. Por lo tanto, cuando se producen algunos eventos, los registros de eventos múltiples se almacenan en diferentes tipos de registro de eventos y ha menudo a través de diferentes archivos de registro de eventos. Por ejemplo, cuando un usuario inicia sesión en un sistema de consola, se registrará un evento en el registro de eventos de seguridad, se registrarán un par de eventos en el registro Microsoft-Windows-TerminalServices-LocalSessionManager/Operational log, and a couple of events will also be recorded in the Microsoft-Windows-TaskScheduler/Operational log.

El registro de eventos también se puede utilizar para detectar si un atacante ha utilizado algún tipo de técnicas anti-forense. Una de esas técnicas sería cambiar el tiempo del sistema para engañar a un investigador. Para detectar este tipo de modificación, un investigador tiene que enumerar todos los registros de registro de eventos disponibles por el número de secuencia y el tiempo generado. Si la hora del sistema se ha revertido, existiría un punto en el que el momento en que se generó un evento era anterior al evento anterior. Algunos ejemplos más de detección de técnicas anti-forense con la ayuda de Windows Event Log se pueden encontrar en una entrada de blog de Harlan Carvey, en http://windowsir.blogspot. de/2013/07/howto-determinedetect-use-of-anti.html.

Analizar el registro de eventos para IOC

Cuando se habla de registros de eventos y se analizan estos registros con Python, no hay forma de evitar **python-evtx**. Estas secuencias de comandos (https://github.com/williballenthin/python-evtx) se han desarrollado utilizando las etiquetas 2.7+ del lenguaje de programación Python. Como es puramente Python, el módulo funciona igualmente bien en las plataformas. El código no depende de ningún módulo que requiera una compilación separada y opera en los archivos de registro de eventos de los sistemas operativos Windows que son más recientes que Windows Vista que es EVTX.

La segunda herramienta que queremos que llama su atención es **plaso**, (refiérase a http://plaso.kiddaland.net/). Este conjunto de herramientas ha evolucionado desde **log2timeline** y ahora está construido en Python. Con la ayuda de este conjunto de herramientas, puede crear cronogramas significativos de los eventos del sistema y otros archivos de registro (por ejemplo, Apache). También hay una muy buena hoja de trucos, http://digital-forensics.sans.org/media/log2timeline_cheatsheet.pdf para log2timeline que demuestra el verdadero poder de esta herramienta. Una de las grandes ventajas de este conjunto de herramientas es el hecho de que incluso se puede ejecutar en una imagen completa de un sistema para generar una línea de tiempo de todas las acciones que los usuarios realizaron en ese sistema antes de crear la imagen.

En las secciones siguientes, mostraremos algunos ejemplos de cómo utilizar **python-evtx** para encontrar el IOC en el registro de eventos de Windows y cómo plaso le ayudará a identificar más IOC y mostrarlos en una línea de tiempo bien formateada.

El parser python-evtx

Antes que nada, queremos comenzar con una conversión básica del formato XML binario de los archivos EVTX a los archivos XML legibles. Esto se puede hacer usando evtxdump.py, https://github.com/williballenthin/python-evtx, que también será la base de nuestros siguientes scripts:

```
#!/usr/bin/env python
import mmap
import contextlib
```

```
import argparse

from Evtx.Evtx import FileHeader
from Evtx.Views import evtx_file_xml_view

def main():
    parser = argparse.ArgumentParser(description="Dump a binary EVTX
file into XML.")
    parser.add_argument("--cleanup", action="store_true",
help="Cleanup unused XML entities (slower)"),
    parser.add_argument("evtx", type=str, help="Path to the Windows
EVTX event log file")
    args = parser.parse_args()

    with open(args.evtx, 'r') as f:
        with contextlib.closing(mmap.mmap(f.fileno(), 0, access=mmap.
ACCESS_READ)) as buf:

            fh = FileHeader(buf, 0x0)
            print "<?xml version=\"1.0\" encoding=\"utf-8\"
standalone=\"yes\" ?>"
            print "<Events>"
            for xml, record in evtx_file_xml_view(fh):
                print xml
            print "</Events>"

if __name__ == "__main__":
    main()
```

Cuando se descarga un evento de inicio de sesión (identificador de evento 4724) con la ayuda del script mencionado anteriormente, el resultado será similar al siguiente:

```
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/
event"><System><Provider Name="Microsoft-Windows-Security-
Auditing" Guid="54849625-5478-4994-a5ba-3e3b0328c30d"></Provider>
<EventID Qualifiers="">4724</EventID>
<Version>0</Version>
<Level>0</Level>
<Task>13824</Task>
<Opcode>0</Opcode>
<Keywords>0x8020000000000000</Keywords>
<TimeCreated SystemTime="2013-11-21 10:40:51.552799"></TimeCreated>
<EventRecordID>115</EventRecordID>
<Correlation ActivityID="" RelatedActivityID=""></Correlation>
<Execution ProcessID="452" ThreadID="1776"></Execution>
<Channel>Security</Channel>
```

```
<Computer>windows</Computer>
<Security UserID=""></Security>
</System>
<EventData><Data Name="TargetUserName">mspreitz</Data>
<Data Name="TargetDomainName">windows</Data>
<Data Name="TargetSid"
>S-1-5-21-3147386740-1191307685-1965871575-1000</Data>
<Data Name="SubjectUserSid">S-1-5-18</Data>
<Data Name="SubjectUserName">WIN-PC9VCSAQB0H$</Data>
<Data Name="SubjectDomainName">WORKGROUP</Data>
<Data Name="SubjectLogonId">0x00000000000003e7</Data>
</EventData>
</Event>
```

Al usar `evtxdump.py`, `https://github.com/williballenthin/python-evtx`, con un archivo de registro de eventos de Windows grande, la salida será muy grande ya que encontrará todos los registros grabados en el archivo XML generado. Para un analista, a menudo es importante realizar una clasificación rápida o buscar eventos específicos rápidamente. Debido a esto, modificamos el script de manera tal que es posible extraer solo eventos específicos, como se muestra acontinuacion:

```
#!/usr/bin/env python
import mmap
import contextlib
import argparse
from xml.dom import minidom

from Evtx.Evtx import FileHeader
from Evtx.Views import evtx_file_xml_view

def main():
    parser = argparse.ArgumentParser(description="Dump specific
event ids from a binary EVTX file into XML.")
    parser.add_argument("--cleanup", action="store_true",
help="Cleanup unused XML entities (slower)"),
    parser.add_argument("evtx", type=str, help="Path to the
Windows EVTX event log file")
    parser.add_argument("out", type=str, help="Path and name of the
output file")
    parser.add_argument("--eventID", type=int, help="Event id
that should be extracted")
    args = parser.parse_args()

    outFile = open(args.out, 'a+')
    with open(args.evtx, 'r') as f:
```

```
        with contextlib.closing(mmap.mmap(f.fileno(), 0, access=mmap.
ACCESS_READ)) as buf:
            fh = FileHeader(buf, 0x0)
            outFile.write("<?xml version=\"1.0\" encoding=\"utf-8\"
standalone=\"yes\" ?>")
            outFile.write("<Events>")
            for xml, record in evtx_file_xml_view(fh):
                xmldoc = minidom.parseString(xml)
                event_id = xmldoc.getElementsByTagName('EventID')[0].
childNodes[0].nodeValue
                if event_id == str(args.eventID):
                    outFile.write(xml)
                else:
                    continue
            outFile.write("</Events>")

if __name__ == "__main__":
    main()
```

Si ahora desea extraer todos los eventos de inicio de sesión del registro de eventos de seguridad de un sistema Windows en un archivo XML dado, solo tiene que ejecutar el script de la siguiente manera:

```
user@lab:~$ ./evtxdump.py security.evtx logon_events.xml –eventID 4724
```

Las herramientas plaso y log2timeline

En esta sección, mostraremos cómo encontrar los eventos de inicio de sesión y cierre de sesión en un Terminal Server. Los eventos de inicio de sesión y cierre de sesión de los Servicios de Terminal Server pueden etiquetarse utilizando `plasm` y filtrarse usando `psort` para obtener una descripción rápida de qué usuarios han iniciado sesión en una máquina y cuándo y dónde. Esta información puede ser muy útil al buscar compromisos. Para comenzar con el plaso, primero necesita etiquetar todos sus datos. Etiquetar con plaso es tan fácil como se muestra acontinuacion:

```
user@lab:~$ ./plasm.py tag --tagfile=tag_windows.txt storage_file
```

Después del etiquetado exitoso, puede buscar en el archivo de almacenamiento las etiquetas con el siguiente comando:

```
user@lab:~$ ./psort.py storage_file "tag contains 'Session logon
succeeded'"
```

El resultado de esta ejecución de comando le mostrará todos los eventos de inicio de sesión exitosos en un sistema determinado. Se pueden ejecutar comandos similares al buscar los servicios que se inician o fallas de EMET.

Ahora que ha visto el tipo de datos que puede extraer del Registro de eventos de Windows, le mostraremos un segundo componente de Microsoft Windows que es realmente útil a la hora de buscar el IOC o al intentar reconstruir el comportamiento del usuario.

Analizando el registro de Windows

El Registro de Windows es uno de los componentes esenciales de los sistemas operativos actuales de Microsoft Windows y, por lo tanto, también es un punto muy importante en una investigación forense. Realiza dos tareas críticas para el sistema operativo Windows. Primero, es el repositorio de configuraciones para el sistema operativo Windows y las aplicaciones que están instaladas en el sistema. Segundo, es la base de datos de la configuración de todo el hardware instalado. Microsoft define el registro de la siguiente manera:

> "*Una base de datos jerárquica central utilizada en Microsoft Windows 98, Windows CE, Windows NT y Windows 2000 utilizada para almacenar información necesaria para configurar el sistema para uno o más usuarios, aplicaciones y dispositivos de hardware*". *(Diccionario de equipos de Microsoft)*

En las siguientes secciones, explicaremos varios elementos del Registro de Windows que pueden ser importantes para un investigador forense y que ayudan a comprender dónde encontrar los indicadores más valiosos. Comenzaremos con una descripción general de la estructura para ayudarlo a encontrar su camino a través de la gran cantidad de datos en el registro. Luego, demostraremos algunos scripts útiles para extraer los **indicadores de compromiso (IOC)**.

Estructura del registro de Windows

En el sistema operativo de Windows, el Registro de Windows se organiza lógicamente en varias claves raíz. Hay cinco claves raíz lógicas en el Registro de Windows de un sistema Windows 7, como se muestra a continuación:



La figura anterior muestra las cinco claves de raíz del Registro en un sistema de Windows 7 que se muestran en el Editor del Registro de Windows (una de las herramientas más comunes para ver y examinar el Registro de Windows).

Hay dos tipos de claves de raíz: volátil y no volátil. Solo hay dos claves raíz que se almacenan en el disco duro del sistema y los datos no volátiles se mantienen en la memoria principal: **HKEY_LOCAL_MACHINE** y **HKEY_USERS**. Las otras claves raíz son los subconjuntos de estas claves o son las claves volátiles que solo se pueden examinar durante el tiempo de ejecución o al descargar la memoria de un sistema antes de comenzar el análisis de su imagen.

El sistema operativo de Windows organiza el Registro en varios archivos de la colmena. De acuerdo con Microsoft,(Referencia `https://msdn.microsoft.com/en-us/library/ windows/desktop/ms724877%28v=vs.85%29.aspx`), la colmena se define de la siguiente manera:

> *Una colmena es un grupo lógico de claves, subclaves y valores en el registro que tiene un conjunto de archivos auxiliares que contienen las copias de seguridad de sus datos.*

Si un nuevo usuario inicia sesión en una máquina con Windows, se crea una sección de perfil de usuario. Esta sección contiene información de registro específica (por ejemplo, configuración de la aplicación, entorno de escritorio, conexiones de red e impresoras) y se encuentra en la clave **HKEY_USERS**.

Cada sección tiene archivos adicionales que se almacenan en el directorio % SystemRoot%\ System32\Config directory. Estos archivos se actualizan cada vez que un usuario inicia sesión y las extensiones de nombre de archivo de estos directorios indican el tipo de datos que contienen. Consulte la siguiente tabla para obtener más detalles (referencia tomada de https://msdn.microsoft.com/en-us/library/windows/desktop/ms724877%28v=vs.85%29.aspx):

Extension	**Descripción**
none	Una copia completa de los datos de la colmena.
`.alt`	Una copia de seguridad de la sección crítica HKEY_LOCAL_MACHINE \System. Solo la clave del sistema tiene un archivo .alt.
`.log`	Un registro de transacciones de cambios en las claves y entradas de valores en la sección.
`.sav`	Una copia de seguridad de una colmena.

En la siguiente sección, analizaremos dónde encontrar interesantes colmenas y cómo analizarlas con la ayuda de las herramientas de Python.

Análisis del registro para el IOC

En esta sección, analizaremos qué colmenas de registro son importantes cuando se busca el IOC. Estas subsecciones incluyen los siguientes temas:

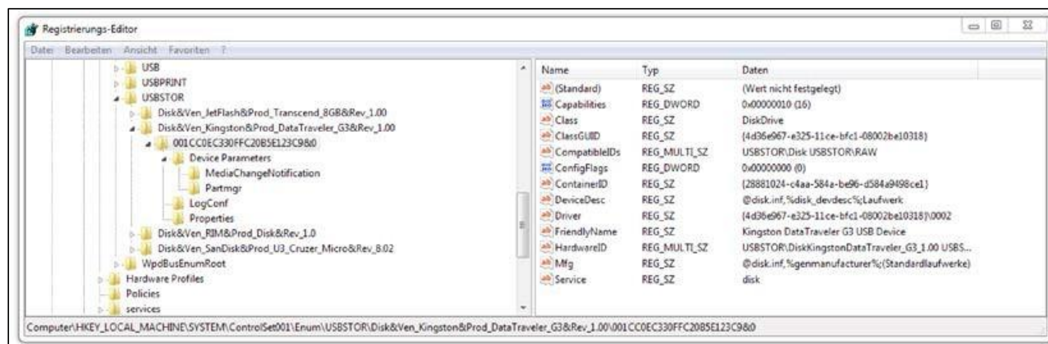
- **Dispositivos USB conectados**: esta sección mostrará qué dispositivos se han conectado a un sistema y cuándo. Esto ayuda a identificar las posibles formas de fuga de datos o exfiltración a través de un usuario del sistema.
- **Historial de usuario**: esta sección mostrará dónde encontrar historiales de archivos abiertos.
- **Programas de inicio**: esta sección mostrará qué programas se ejecutarán al inicio del sistema. Esto puede ser muy útil cuando se trata de identificar los sistemas infectados.
- I**nformación del sistema**: esta sección mostrará dónde encontrar información importante del sistema en cuestión (por ejemplo, nombres de usuario).
- **Shim Cache Parser**: esta sección mostrará cómo obtener un IOC importante con la ayuda de herramientas comunes de Python, como **Shim Cache Parser** de Mandiant, consulte `https://www.mandiant.com/blog/leveraging-application-compatibility-cache-forensic-investigations/`.

Dispositivos USB conectados

Una de las preguntas más comunes que una persona de respuesta a incidentes debe responder es si un usuario ha extraído datos confidenciales de un sistema o si un compromiso del sistema ha sido iniciado por un dispositivo USB deshonesto que un usuario conectado al sistema. Para responder a esta pregunta, el registro de Windows es un buen punto para comenzar.

Cada vez que se conecta un nuevo dispositivo USB al sistema, dejará información en el registro. Esta información puede identificar de forma única para cada dispositivo USB que se haya conectado al sistema. El registro almacena la identificación del proveedor, la identificación del producto, la revisión y los números de serie de cada dispositivo USB conectado. Esta información se puede encontrar en la sección de registro,

`HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Enum\USBSTOR` , *Windows Forensic Analysis*, *Harlan Carvey*, *Dave Kleiman*, *Syngress Publishing*, que también se muestra en la siguiente captura de pantalla:



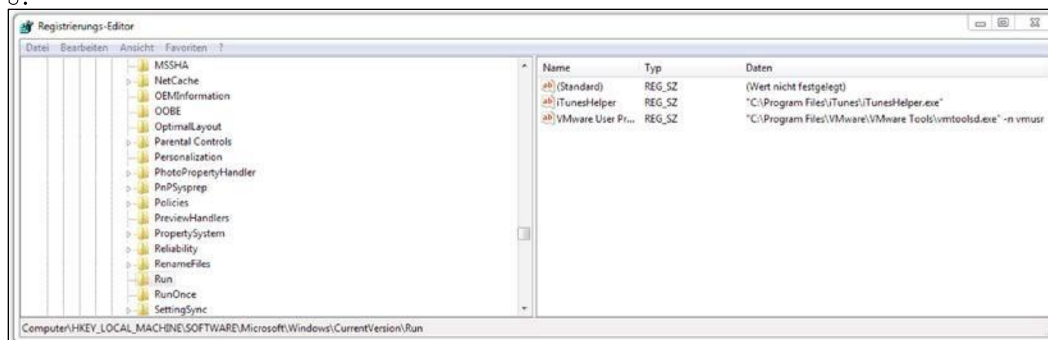
Historial de usuarios

En un sistema Windows, hay varias listas en el Registro que ayudan a identificar la actividad reciente del usuario (por ejemplo, páginas web recientemente visitadas o archivos de Microsoft Word recientemente abiertos). La siguiente tabla muestra algunas de estas listas con las correspondientes subclaves del Registro de Windows, para todas las listas y sus subclaves del Registro de Windows, consulte `http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1071&context=adf`:

Lista de historial	**Clave de registro de Windows relacionada**
URL tipeadas en Microsoft Internet Explorer	HKEY_USERS\S-1-5-21-[User Identifier] \Software\Microsoft\Internet Explorer\TypedURLs
Los archivos de Microsoft Office usados más recientemente	HKEY_USERS\S-1-5-21-[User Identifier] \Software \Microsoft\Office\12.0\Office_App_Name\File MRU
Las unidades de red asignadas más recientemente	HKEY_USERS\S-1-5-21-[User Identifier] \Software \Microsoft\Windows\CurrentVersion\Explorer\Map Network Drive MRU
El comando más reciente escrito en el cuadro de diálogo RUN	HKEY_USERS\S-1-5-21-[User Identifier] \Software \Microsoft\Windows\CurrentVersion\Explorer\RunMRU
Carpetas recientes	HKEY_USERS\S-1-5-21-[User Identifier] \Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDocs\Folder

Programas de inicio

Durante algunas investigaciones, es importante saber qué software se ejecutó automáticamente al inicio y qué software se inició manualmente por un usuario. Para ayudar a responder esta pregunta, el Registro de Windows `HKEY_LOCAL_MACHINE \SOFTWARE\ Microsoft\Windows\CurrentVersion\Run` puede volver a ayudar. La lista de programas de inicio se muestra en la siguiente figura y se enumera en `Windows Register hive`, que se toma `de Windows Registry Quick Reference, Farmer, D. J`:



Información del sistema

En esta sección, veremos algunas colmenas de registro que pueden ser importantes al analizar un sistema. En primer lugar, hay una gran cantidad de información sobre la cuenta de usuario que se almacena en el registro, de la siguiente manera:

- Una lista de cuentas de usuario
- Última hora de inicio de sesión de cada cuenta
- Si la cuenta requiere una contraseña
- Si una cuenta específica está deshabilitada o habilitada.
- La técnica de hash que se usa para calcular el hash de la contraseña

Toda esta información se encuentra en la siguiente clave del registro:

```
HKEY_LOCAL_MACHINE\SAM\Domains\Account\Users
```

Hay muchos más datos interesantes en el Registro de Windows; sin embargo, un tipo de información puede ser muy útil durante una investigación forense: la hora de la última parada del sistema. Esta información se almacena en el valor `ShutdownTime` en la siguiente sección:

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Windows
```

Esta información es a menudo interesante en los sistemas de servidor, ya que podría ser una pista sobre cuándo se aplicaron las últimas actualizaciones o si hubo reinicios no planificados de un sistema, lo que también podría haber sido causado por un atacante.

Shim Cache Parser

El Registro de Windows contiene problemas de compatibilidad de aplicaciones y una gran cantidad de metadatos (como el tamaño del archivo, la última vez que se modificó el archivo y el último tiempo de ejecución según la versión del sistema operativo) que podría ser importante para el tiempo de ejecución de la aplicación en **Application Compatibility Shim Cache**.

> La implementación y la estructura de la caché Shim de compatibilidad de aplicaciones pueden variar según el sistema operativo. Por lo tanto, verifique sus hallazgos a fondo.

Los datos sobre la compatibilidad de la aplicación y los problemas de tiempo de ejecución pueden ser muy útiles durante una respuesta de incidente o cualquier otro tipo de investigación forense para identificar los sistemas potencialmente infectados y crear una línea de tiempo de cuándo tuvo lugar la infección potencial. Mandiant ha lanzado una herramienta para extraer este tipo de evidencia: **Shim Cache Parser** (para obtener más información, consulte https://github.com/mandiant/ShimCacheParser)

> *Shim Cache Parser determinará automáticamente el formato de los datos almacenados en caché y emitirá su contenido. Admite una serie de entradas, incluidas las inscripciones en el registro del sistema, el binario sin procesar o el registro del sistema actual.*

La herramienta se puede usar contra una sección del registro exportada o contra el sistema en ejecución. Cuando lo use contra un sistema en ejecución, solo necesita ejecutar el siguiente comando:

```
C:\tools\mandiant> python ShimCacheParser.py –l –o out.csv

[+] Dumping Shim Cache data from the current system...
[+] Found 64bit Windows 7/2k8-R2 Shim Cache data...
[+] Found 64bit Windows 7/2k8-R2 Shim Cache data...
[+] Found 64bit Windows 7/2k8-R2 Shim Cache data...
[+] Writing output to out.csv...
```

Al mirar la salida CSV generada, puede encontrar las aplicaciones instaladas y el primer tiempo de ejecución de estos archivos, de la siguiente manera:

```
Last Modified,Last Update,Path,File Size,Exec Flag
05/04/11 05:19:28,N/A,C:\Windows\system32\SearchFilterHost.exe,N/A,True
05/24/15 16:44:45,N/A,C:\Program Files (x86)\Avira\AntiVir
Desktop\avwsc. exe,N/A,True
11/21/10 03:24:15,N/A,C:\Windows\system32\wbem\wmiprvse.exe,N/A,True
05/30/14 08:07:49,N/A,C:\Windows\TEMP\40F00A21-D2E7-47A3-AE16-
0AFB8E6C1F87\dismhost.exe,N/A,True
07/14/09 01:39:02,N/A,C:\Windows\system32\DeviceDisplayObjectProvider.
exe,N/A,True
07/26/13 02:24:56,N/A,C:\Windows\System32\shdocvw.dll,N/A,False
05/24/15 16:46:22,N/A,C:\Program Files
(x86)\Google\Update\1.3.27.5\ GoogleCrashHandler.exe,N/A,True
05/07/15 21:42:59,N/A,C:\Windows\system32\GWX\GWX.exe,N/A,True
03/26/15 20:57:08,N/A,C:\Program Files (x86)\Parallels\Parallels
Tools\ prl_cc.exe,N/A,True
10/07/14 16:29:54,N/A,C:\Program Files (x86)\KeePass Password Safe
2\ KeePass.exe,N/A,True
10/07/14 16:44:13,N/A,C:\ProgramData\Avira\Antivirus\TEMP\SELFUPDATE\
updrgui.exe,N/A,True
04/17/15 21:03:48,N/A,C:\Program Files (x86)\Avira\AntiVir
Desktop\ avwebg7.exe,N/A,True
```

Mirando los datos anteriores, uno puede ver que el usuario instaló o actualizó Avira AntiVir el 24-05-2015 y KeePass el 2014-07-10. Además, puede encontrar algunas sugerencias de que el sistema parece ser un sistema virtual, ya que puede ver las sugerencias de Parallels, una plataforma de virtualización Mac OS X.

Si se consideran las herramientas que se han descrito anteriormente y la información que contienen el registro de eventos de Windows y el registro de Windows, está claro que en una investigación forense, no todas las preguntas relativas a un sistema pueden responderse sin estas fuentes de información.

Implementación de verificaciones específicas de Linux

En esta sección, describiremos cómo implementar algunas comprobaciones de integridad para respaldar los signos de búsqueda de la manipulación del sistema en Linux y sistemas similares (por ejemplo, BSD).

Estas comprobaciones incluyen lo siguiente:

- Búsqueda de anomalías en la gestión de usuarios locales.
- Comprender y analizar metadatos de archivos para permisos y privilegios especiales.
- Usar algoritmos de agrupación en metadatos de archivo para obtener indicadores sobre dónde buscar más profundo

Comprobación de la integridad de las credenciales de usuario locales

La información sobre los usuarios locales en Linux se almacena principalmente en dos archivos: /etc/passwd y /etc/shadow. Este último es opcional y toda la información sobre los usuarios locales, incluida la contraseña hash, se almacenó originalmente en / etc/passwd. Pronto, se consideró un problema de seguridad para almacenar la información de la contraseña en un archivo que es legible para cada usuario. Por lo tanto, los hashes de contraseña en /etc/passwd fueron reemplazados por una sola x que indica que el hash de la contraseña correspondiente debe buscarse en /etc/shadow.

El efecto secundario de este proceso evolutivo es que los hash de contraseña en / /etc/ passwd todavía son compatibles y todas las configuraciones en /etc/passwd pueden anular las credenciales en /etc/shadow.

Ambos archivos son archivos de texto con una entrada por línea. Una entrada consta de varios campos separados por dos puntos.

El formato de /etc/passwd es el siguiente:

- **username**: Este campo contiene el nombre de usuario legible por humanos. No es necesario que el nombre de usuario sea único. Sin embargo, la mayoría de las herramientas de administración de usuarios imponen nombres de usuario únicos.
- **password hash**: Este campo contiene la contraseña en una forma codificada de acuerdo con la función Posix `crypt()` Si este campo está vacío, el usuario correspondiente no necesita una contraseña para iniciar sesión en el sistema. Si este campo contiene un valor que no puede ser generado por el algoritmo hash, por ejemplo, un signo de exclamación, entonces el usuario no puede iniciar sesión usando una contraseña. Sin embargo, esta condición no hace que la cuenta sea inútil. Un usuario con una contraseña bloqueada todavía puede iniciar sesión utilizando otros mecanismos de autenticación, por ejemplo, claves SSH.

  Como se mencionó anteriormente, el valor especial x significa que el hash de la contraseña debe encontrarse en el archivo shadow.

Comenzando con la biblioteca del sistema glibc2, la función crypt() admite múltiples algoritmos hash. En ese caso, el hash de la contraseña tiene el siguiente formato:

```
$id$salt$encrypted
```

El ID designa el algoritmo hash que se ha utilizado para codificar la contraseña, por ejemplo, 1 para md5, 5 para sha256 y 6 para sha512. La salt es una cadena generada aleatoriamente para modificar el algoritmo hash. En consecuencia, incluso las contraseñas idénticas producen diferentes sumas de hash. El subcampo "encriptado" contiene el hash real de la contraseña (modificado por la influencia de la salt).

- **numerical user ID**: Este campo denota la identificación del usuario. Internamente, el kernel de Linux usa solo este ID numérico. El ID especial 0 se asigna al usuario raíz administrativo. Por defecto, el ID de usuario 0 recibe privilegios ilimitados en el sistema.
- **numerical group ID**: Este campo se refiere al grupo primario del usuario.
- **comment field**: Este campo puede contener información arbitraria sobre el usuario y se utiliza principalmente para contener el nombre completo del usuario. A veces, también contiene una lista separada por comas del nombre de usuario completo, número de teléfono, etc.
- **user home directory**: El directorio principal del usuario es un directorio en el sistema de archivos del sistema. Después de iniciar sesión, se inician nuevos procesos con este directorio como directorio de trabajo.
- **default command shell**: Este campo opcional denota el shell predeterminado que se debe iniciar después de un inicio de sesión exitoso.

El formato de /etc/shadowes el siguiente:

- El campo **username** fvincula la entrada a la entrada `passwd` con el mismo nombre de usuario.
- El campo **password hash** contiene la contraseña codificada en el mismo formato que se describe para el archivo `passwd`.
- Los próximos cinco campos contienen información sobre el envejecimiento de la contraseña, como la fecha del último cambio de contraseña, la edad mínima de la contraseña, la edad máxima de la contraseña, el período de advertencia de la contraseña y el período de inactividad de la contraseña.
- Si el campo **account expiration date** no está vacío, se interpretará como la fecha de vencimiento de la cuenta. Esta fecha se expresa en días desde el 1 de enero de 1970.

Con esta descripción de formato, una pequeña rutina de Python es suficiente para analizar el archivo en una lista de entradas, cada una con una lista de campos como se muestra a continuación:

```
def read_passwd(filename):
    """Reads entries from shadow or passwd files and
       returns the content as list of entries.
       Every entry is a list of fields."""

    content = []
    with open(filename, 'r') as f:
        for line in f:
            entry = line.strip().split(':')
            content.append(entry)

    return content
```

Al usar esta rutina, pueden detectarse manipulaciones típicas en estos archivos.

La primera técnica de manipulación que queremos describir es la `creación de múltiples usuarios que comparten el mismo ID numérico`. Esta técnica puede ser utilizada por los atacantes para plantar una puerta trasera en el sistema. Al crear un usuario adicional para una ID existente, un atacante puede crear un alias con una contraseña independiente. El propietario legítimo de la cuenta no sabría que hay una combinación adicional de nombre de usuario/contraseña para iniciar sesión en la cuenta.

Una pequeña rutina de Python puede detectar este tipo de manipulación, de la siguiente manera:

```
def detect_aliases(passwd):
    """Prints users who share a user id on the console

       Arguments:
       passwd -- contents of /etc/passwd as read by read_passwd"""

    id2user = {}
    for entry in passwd:
        username = entry[0]
        uid = entry[2]
        if uid in id2user:
            print 'User "%s" is an alias for "%s" with uid=%s' %
(username, id2user[uid], uid)
        else:
            id2user[uid] = username
```

Durante el funcionamiento normal, la información en /etc/passwd y /etc/shadow se sincroniza, es decir, cada usuario debe aparecer en ambos archivos. Si hay usuarios que aparecen en solo uno de estos archivos, es un indicador de que se ha omitido la administración de usuarios del sistema operativo. Una manipulación como esta se puede detectar con un script similar:

```
def detect_missing_users(passwd, shadow):
    """Prints users of /etc/passwd missing in /etc/shadow
       and vice versa.

       Arguments:
       passwd -- contents of /etc/passwd as read by read_passwd
       shadow -- contents of /etc/shadow as read by
       read_passwd"""

    passwd_users = set([e[0] for e in passwd])
    shadow_users = set([e[0] for e in shadow])

    missing_in_passwd = shadow_users - passwd_users
    if len(missing_in_passwd) > 0:
        print 'Users missing in passwd: %s' % ', '.join(missing_in_
passwd)

    missing_in_shadow = passwd_users - shadow_users
    if len(missing_in_shadow) > 0:
        print 'Users missing in shadow: %s' % ', '.join(missing_in_
shadow)
```

Al igual que con la primera función, esta función no debe producir ningún resultado en un sistema normal. Si hay una salida similar a los usuarios que faltan en shadow: backdoor, entonces hay una cuenta de usuario "puerta trasera" en el sistema sin un registro en el archivo shadow.

Los usuarios sin contraseña no deberían existir en un sistema normal. Además, todos los hashes de contraseña deben residir en el archivo shadow y todas las entradas en el archivo de passwd deben referirse a la entrada de shadow correspondiente. El siguiente script detecta desviaciones de esta regla:

```
def detect_unshadowed(passwd, shadow):
    """Prints users who are not using shadowing or have no password
set

       Arguments:
       passwd -- contents of /etc/passwd as read by read_passwd
       shadow -- contents of /etc/shadow as read by read_passwd"""


    nopass = [e[0] for e in passwd if e[1]=='']
```

```
        nopass.extend([e[0] for e in shadow if e[1]==''])
        if len(nopass) > 0:
            print 'Users without password: %s' % ', '.join(nopass)

        unshadowed = [e[0] for e in passwd if e[1] != 'x' and e[1] != '']
        if len(unshadowed) > 0:
            print 'Users not using password-shadowing: %s' % \
                  ', '.join(unshadowed)
```

Nuestro último ejemplo de omitir el sistema operativo en la creación y manipulación de cuentas de usuario es la detección de algoritmos hash no estándar y la reutilización de **salts** para múltiples cuentas de usuario. Si bien un sistema Linux permite especificar el algoritmo de hash para cada entrada en el archivo shadow, normalmente todas las contraseñas de usuario son hash usando el mismo algoritmo. Un algoritmo de desviación es una señal para que una entrada se escriba en el archivo shadow sin utilizar las herramientas del sistema operativo, es decir, la manipulación del sistema. Si se reutiliza una salts a través de múltiples hashes de contraseñas, entonces el **salts** se codifica en una herramienta de manipulación o las rutinas criptográficas del sistema se han visto comprometidas, por ejemplo, manipulando la fuente de entropía de la generación de salts.

El siguiente script de Python es capaz de detectar este tipo de manipulación:

```
import re
def detect_deviating_hashing(shadow):
    """Prints users with non-standard hash methods for passwords

       Arguments:
       shadow -- contents of /etc/shadow as read by read_passwd"""

    noalgo = set()
    salt2user = {}
    algorithms = set()
    for entry in shadow:
        pwhash = entry[1]
        if len(pwhash) < 3:
            continue

        m = re.search(r'^\$([^$]{1,2})\$([^$]+)\$', pwhash)
        if not m:
            noalgo.add(entry[0])
            continue

        algo = m.group(1)
        salt = m.group(2)

        if salt in salt2user:
```

```
            print 'Users "%s" and "%s" share same password salt "%s"'
% \
                  (salt2user[salt], entry[0], salt)
        else:
            salt2user[salt] = entry[0]

        algorithms.add(algo)

    if len(algorithms) > 1:
        print 'Multiple hashing algorithms found: %s' % ',
'.join(algorithms)

    if len(noalgo) > 0:
        print 'Users without hash algorithm spec. found: %s' % \
              ', '.join(noalgo)
```



Expresiones regulares

El último ejemplo usa el re módulo para la coincidencia de expresión regular para extraer la especificación del algoritmo y la salt del hash de la contraseña. Las expresiones regulares proporcionan una manera rápida y poderosa de buscar, emparejar, dividir y reemplazar texto. Por lo tanto, le recomendamos familiarizarse con las expresiones regulares. La documentación del módulo re está disponible en línea en https://docs.python.org/2/library/re.html. El libro Mastering Python Regular Expressions, Felix Lopez y Victor Romero, Packt Publishing ofrece más información y ejemplos sobre cómo usar expresiones regulares.

Todos los métodos de detección en esta sección son ejemplos de métodos de detección de anomalías. Según el entorno del sistema, se pueden usar e implementar detecciones de anomalías más específicas siguiendo el esquema de los ejemplos. Por ejemplo, en un sistema de servidor, el número de usuarios que tienen un conjunto de contraseña debe ser pequeño. Por lo tanto, contar a todos los usuarios con contraseñas puede ser un paso razonable en el análisis de dichos sistemas.

Análisis de la meta información de archivo

En esta sección, discutiremos la metainformación de archivo y proporcionaremos ejemplos de cómo se puede usar en el análisis forense.

Comprensión del inodo

Los sistemas Linux almacenan metadatos de archivo en estructuras llamadas **inodos (nodos índice).** En un sistema de archivos de Linux, cada objeto está representado por un inodo. Los datos almacenados por inodo dependen del tipo de sistema de archivos real. Los contenidos típicos de un inodo son los siguientes:

- El **número de índice** es el identificador de un inodo. El número de índice es único por sistema de archivos. Si dos archivos comparten el mismo número de índice, estos archivos están **vinculados**. En consecuencia, los archivos vinculados solo difieren en su nombre de archivo y siempre tienen los mismos contenidos, así como la misma metainformación.
- El **propietario del archivo** está definido por la ID numérica del usuario (UID). Solo puede haber un propietario por archivo. Las ID de usuario deben corresponder a las entradas en /etc/passwd. Sin embargo, no se garantiza que solo haya archivos con entradas existentes en /etc/passwd. Los archivos se pueden transferir a los usuarios que no tienen privilegios administrativos. Además, es posible que el propietario del archivo se haya eliminado del sistema, lo que hizo que el archivo quedara huérfano. Para los archivos en medios transportables, por ejemplo, unidades USB, no existe un mecanismo para asignar la ID de usuario de un sistema a otro. En consecuencia, el propietario del archivo parece cambiar cuando se conecta un disco USB a un nuevo sistema con diferentes /etc/passwd. Además, esto también puede generar archivos huérfanos si no existe un UID en el sistema donde está conectado el disco USB.
- El **grupo de archivos** está definido por la ID numérica del grupo correspondiente (GID). Un archivo siempre se asigna a un grupo exactamente. Todos los grupos de un sistema deben definirse en /etc/groups. Sin embargo, pueden existir archivos con ID de grupo que no figuran en /etc/groups. Esto indica que el grupo correspondiente se ha eliminado del sistema, que el medio se ha transferido desde otro sistema donde existe ese grupo o que un usuario con privilegios administrativos reasignó el archivo a un grupo inexistente.
- El **modo de archivo (también conocido como "bits de protección")** define una forma simple de derechos de acceso al archivo correspondiente. Es una máscara de bits que define los derechos de acceso para el propietario del archivo, para los usuarios que pertenecen al grupo al que está asignado el archivo y para todos los demás usuarios. Para cada uno de estos casos, se definen los siguientes bits:
    - **read** (**r**): Si este bit se establece en un archivo normal, el usuario afectado puede leer el contenido del archivo. Si el bit se establece en un directorio, el usuario afectado puede mostrar los nombres del contenido del directorio. El acceso de lectura no incluye la metainformación, que es la información de inodo de las entradas del directorio. En consecuencia, el permiso de lectura para un directorio no es suficiente para leer archivos en ese directorio, ya que esto requeriría acceso a los datos de inodo del archivo.

    - **write** (**w**): Si este bit se establece en un archivo normal, el usuario afectado puede modificar el contenido del archivo de forma arbitraria, incluida la manipulación y eliminación del contenido. Si este bit se establece en una entrada de directorio, entonces el usuario afectado puede crear, eliminar y cambiar el nombre de las entradas en ese directorio. Los archivos existentes en el directorio tienen sus propios bits de protección que definen sus derechos de acceso.
    - **execute** (**x**): Para los archivos normales, esto permite que el usuario afectado inicie el archivo como un programa. Si el archivo es un binario compilado, por ejemplo, en formato ELF, entonces los privilegios de ejecución son suficientes para ejecutar el programa. Si el archivo es un script que debe interpretarse, entonces también se requiere permiso de lectura (r) para ejecutar el script. La razón es que el kernel de Linux determina cómo cargar el programa. Si detecta que el archivo contiene una secuencia de comandos, carga el intérprete de secuencia de comandos con los privilegios del usuario actual. Para los directorios, este indicador otorga permiso para leer la metainformación del contenido del directorio, excepto los nombres de las entradas. Por lo tanto, esto permite que el usuario afectado cambie el directorio de trabajo a este directorio.
    - **sticky** (**t**): Este bit solo existe una vez por inodo. Cuando se establece en directorios, limita el derecho de eliminar y cambiar el nombre de las entradas al usuario que posee la entrada. En los archivos normales, este indicador se ignora o tiene un efecto específico del sistema de archivos. Cuando se establece en ejecutables, este indicador se usa para evitar que el proceso resultante se intercambie desde la RAM. Sin embargo, este propósito del bit sticky está en desuso y los sistemas Linux no obedecen el bit sticky en los ejecutables.
    - **set id on execution** (**s**): Este bit existe para el usuario y para el grupo. Cuando se establece para el usuario (bit SUID) en un archivo ejecutable, el archivo correspondiente se ejecuta siempre con su propietario como usuario efectivo. Por lo tanto, el programa se ejecuta con los privilegios del usuario que posee el ejecutable que es independiente del usuario que realmente está iniciando el programa. Si el archivo es propiedad del usuario raíz (UID 0), entonces el archivo ejecutable siempre se ejecuta con privilegios ilimitados. Cuando el bit se establece para el grupo (SGID bit), el archivo ejecutable siempre se inicia con el grupo del archivo como grupo efectivo.

- El tamaño del archivo en bytes.
- La cantidad de bloques asignados para ese archivo.
- Una marca de tiempo que denota el último cambio del contenido del archivo (**mtime**).

- Una marca de tiempo que denota el último acceso de lectura al contenido del archivo (**atime**). El acceso a la marca de tiempo se puede deshabilitar mediante la opción de **noatime** para limitar el acceso de escritura a los medios (por ejemplo, para extender la vida útil de las tarjetas SD). Además, el acceso de solo lectura (mount option ro) en el sistema de archivos evita el seguimiento de atime. Por lo tanto, antes del análisis de la información de atime, debería verificarse si el seguimiento de atime estaba habilitado para ese sistema de archivos. Las opciones de montaje inicial correspondientes se pueden encontrar en /etc/fstab.
- Una marca de tiempo que denota el último cambio de los datos de inodo (**ctime**).

Las extensiones notables a estas entradas estándar son **listas de control de acceso de POSIX (POSIX ACLs)**. Estas listas de control de acceso son compatibles con los principales sistemas de archivos de Linux y permiten especificar entradas de acceso adicionales además de las tres clases (usuario, grupo y otros). Estas entradas permiten definir los derechos de acceso adicionales (los bits r, w y x listados anteriormente) para usuarios y grupos adicionales. La evaluación de POSIX ACL se discutirá en detalle en una sección separada.

Otra extensión consiste en la especificación de **indicadores de capacidad** para un ejecutable. Esto se usa para una especificación de privilegios más granular que el uso del bit SUID. En lugar de proporcionar un ejecutable propiedad del usuario root con el bit SUID y permitirle privilegios ilimitados, se puede especificar un conjunto de privilegios necesarios. Las capacidades también se tratarán en detalle en una sección separada.

Lectura de metadatos de archivos básicos con Python

Python proporciona funcionalidad incorporada para leer la información del estado del archivo con el módulo `os`. La función estándar para recuperar metadatos de un archivo especificado por su nombre es `os.lstat()`. En contraste con el `os.stat()` más comúnmente utilizado, esta función no evalúa los objetivos de los enlaces simbólicos, sino que recupera la información sobre el enlace en sí. Por lo tanto, no es propenso a correr en bucles infinitos que son causados por enlaces simbólicos circulares. Además, no causa ningún error en los enlaces que carecen del destino del enlace.

El objeto resultante depende de la plataforma; Sin embargo, la siguiente informaciónestá siempre disponible: `st_mode` (bits de protección), `st_ino` (número de inodo), `st_dev` (identificador del dispositivo que contiene el objeto del sistema de archivos), `st_nlink` (número de enlaces duros), `st_uid` (ID de usuario del propietario), `st_gid` (ID de grupo propietario), `st_size` (tamaño de archivo en bytes), `st_mtime` (última modificación), `st_atime` (último acceso de lectura), `st_ctime` (último cambio de inodo). Esta información corresponde a los datos de inodo que se describen en la sección anterior.



Una descripción detallada de os.stat () y os.lstat () está disponible en https://docs.python.org/2/library/os.html#os.stat. Esto también incluye los ejemplos de atributos dependientes de la plataforma.

Las marcas de tiempo `st_mtime`, `st_atime` y `st_ctime` se especifican en el formato de marca de tiempo Unix, es decir, el número de segundos desde el 1 de enero de 1970. Con el módulo datetime, este formato de tiempo se puede convertir en una forma legible para el ser humano utilizando el siguiente script :

```
from datetime import datetime as dt
from os import lstat

stat_info = lstat('/etc/passwd')

atime = dt.utcfromtimestamp(stat_info.st_atime)
mtime = dt.utcfromtimestamp(stat_info.st_mtime)
ctime = dt.utcfromtimestamp(stat_info.st_ctime)

print 'File mode bits:      %s' % oct(stat_info.st_mode)
print 'Inode number:        %d' % stat_info.st_ino
print '# of hard links:     %d' % stat_info.st_nlink
print 'Owner UID:           %d' % stat_info.st_uid
print 'Group GID:           %d' % stat_info.st_gid
print 'File size (bytes)    %d' % stat_info.st_size
print 'Last read (atime)    %s' % atime.isoformat(' ')
print 'Last write (mtime)   %s' % mtime.isoformat(' ')
print 'Inode change (ctime) %s' % ctime.isoformat(' ')
```

Esta lista de códigos produce los valores de retorno comunes de la llamada `lstat`. Un resultado típico es similar al siguiente:

```
File mode bits:      0100644
Inode number:        1054080
# of hard links:     1
Owner UID:           0
Group GID:           0
File size (bytes)    2272
Last read (atime)    2015-05-15 09:25:15.991190
Last write (mtime)   2014-09-20 10:40:46.389162
Inode change (ctime) 2014-09-20 10:40:46.393162
```

Este resultado muestra que en el sistema de laboratorio, `/etc/passwd` es un archivo normal con permiso de lectura para todos los usuarios. Esta información se deriva del miembro `st_mode` del resultado. Al usar la función `oct()` de Python, se convierte en su representación octal, es decir, *un dígito decimal de la salida representa exactamente tres bits de los bits de protección*. El cero inicial en la salida es un indicador común para la representación octal.

Los tres dígitos más bajos (644 en el ejemplo de salida) siempre denotan los derechos de acceso para el propietario del archivo (6 en el ejemplo), para los usuarios que pertenecen al grupo del archivo (izquierda 4 en el ejemplo) y todos los demás usuarios (último dígito).



¿Cómo interpretar los bits del modo de archivo?

En su forma octal, los valores de bits de los tres dígitos menos significativos representan los derechos de acceso para el propietario, grupo y otros usuarios (último dígito). Para cada dígito, el acceso de lectura (r) tiene un valor de bit 4, el acceso de escritura (w) tiene un valor de bit 2 y la ejecución (x) tiene un bit de valor 1.

Por lo tanto, en nuestro ejemplo, el dígito 6 denota acceso de lectura y escritura (4 + 2) para el propietario del archivo. Los miembros del grupo 0 y todos los demás usuarios solo tienen acceso de lectura (4).

El siguiente dígito de la derecha denota el bit sticky (valor 1), el bit SGID (valor 2) y el bit SUID (valor 4).



El módulo `stat` define las constantes para todos los bits de `st_mode`. Su documentación está disponible en https://docs.python.org/2/library/stat.html.

Estas constantes se pueden usar como una máscara de bits para recuperar información de `st_mode`. El ejemplo anterior podría ampliarse para detectar SGID, SUID y modo de conexión, de la siguiente manera:

```
import stat

if stat.S_ISUID & stat_info.st_mode:
    print 'SUID mode set!'

if stat.S_ISGID & stat_info.st_mode:
    print 'SGID mode set!'

if stat.S_ISVTX & stat_info.st_mode:
    print 'Sticky mode set!'
```

Para probar el código, puede usar el ejemplo para evaluar el modo de `/etc/passwd, /tmp, y /usr/bin/sudo` en un sistema Linux estándar. Normalmente, /tmp tiene el conjunto de banderas sticky, /usr/bin/sudo tiene SUID configurado y /etc/password no tiene ninguno de los bits especiales configurados.

Los bits restantes indican el tipo de archivo. Los siguientes tipos de archivos pueden aparecer en un sistema de archivos Linux estándar:

Tipo de archivo	**Comprobar la función en** `stat`	**Descripción**
regular	S_ISREG()	Esto se usa para almacenar datos arbitrarios.
directorio	S_ISDIR()	Esto se usa para almacenar listas de otros archivos.
enlace suave	S_ISLNK()	Esto hace referencia a un archivo de destino a través del nombre
dispositivo de caracteres	S_ISCHR()	Esta es la interfaz en el sistema de archivos para acceder al hardware orientado al carácter, por ejemplo, terminales
dispositivo de bloqueo	S_ISBLK()	Esta es la interfaz en el sistema de archivos para acceder al hardware orientado a bloques, por ejemplo, particiones de disco
fifo	S_ISFIFO()	Esta es la representación de un named, interfaz de interproceso unidireccional en el sistema de archivos
socket	S_ISSOCK()	Esta es la representación de un named, interfaz de interproceso unidireccional en el sistema de archivos

Los enlaces duros no están representados por un tipo de archivo especial, sino que son simplemente varias entradas de directorio en el mismo sistema de archivos que comparten el mismo inodo.

A diferencia de las pruebas para SGID, SUID y bit sticky, las comprobaciones de tipo de archivo se implementan como funciones del módulo `stat`. Estas funciones requieren los bits del modo de archivo como el parámetro, por ejemplo:

```
from os import readlink,lstat
import stat

path = '/etc/rc5.d/S99rc.local'

stat_info = lstat(path)

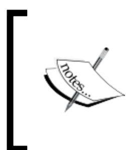
if stat.S_ISREG(stat_info.st_mode):
    print 'File type: regular file'

if stat.S_ISDIR(stat_info.st_mode):
```

```
        print 'File type: directory'

    if stat.S_ISLNK(stat_info.st_mode):
        print 'File type: symbolic link pointing to ',
        print readlink(path)
```

En este ejemplo, la función `os.readlink ()` se utiliza para extraer el nombre del archivo de destino si se encuentra un enlace simbólico. Los enlaces simbólicos pueden referirse a una ruta absoluta o una ruta relativa que comienza desde la ubicación del enlace simbólico en el sistema de archivos. Los enlaces simbólicos absolutos tienen un objetivo que comienza con el carácter /, es decir, el objetivo debe buscarse comenzando por el directorio raíz del sistema.



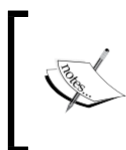
Si monta su copia de la evidencia en su entorno de laboratorio para su análisis, los enlaces simbólicos absolutos o bien se rompen o se apuntan a un archivo en su estación de trabajo de laboratorio! Los enlaces simbólicos relativos permanecen intactos siempre que su destino se encuentre en la misma partición que el enlace.

Un posible resultado del código de ejemplo anterior podría ser - Tipo de archivo: enlace simbólico que apunta a ../init.d/rc.local -, que es un ejemplo de un enlace relativo.

Evaluación de POSIX ACL con Python

Los bits de modo de archivo, que se definen en el inodo del archivo, solo permiten tres destinatarios para los permisos: el propietario del archivo, los usuarios que pertenecen al grupo del archivo y todos los demás.

Si se requiere un conjunto de permisos más granular, la solución tradicional será crear un grupo que esté formado por todos los usuarios que deberían tener acceso y transferir el archivo a ese grupo. Sin embargo, la creación de tales grupos tiene grandes desventajas. Primero, la lista de grupos puede llegar a ser innecesariamente grande. Segundo, la creación de tales grupos requiere privilegios administrativos y, por lo tanto, rompe el concepto Linux/Unix de c**ontrol de acceso discrecional.**



El control de acceso discrecional es el concepto de permitir que el propietario de la información, es decir, el propietario del archivo, decida a quién se debe permitir el acceso. En el control de acceso discrecional, la propiedad es el único requisito para que se le permita conceder o revocar el acceso a un recurso.

Por último, los propietarios de archivos solo pueden abrir archivos y directorios para todos en el sistema si no hay un grupo que coincida con la lista de usuarios a los que autorizar. Esto rompe el concepto de *privilegio mínimo,* es decir, no otorga más permisos en un sistema de los requeridos para su operación.

Para mantener el control de acceso discrecional, así como el concepto de privilegio mínimo, se especificó una extensión opcional al modo de acceso a archivos, es decir, **POSIX ACL**. Además de permitir permisos de lectura, escritura y ejecución para el propietario del archivo, el grupo y otros, las POSIX ACLs permiten especificar lo siguiente:

- Permisos específicos de lectura, escritura y ejecución para usuarios arbitrarios
- Permisos específicos de lectura, escritura y ejecución para grupos arbitrarios.
- No se concede ningún privilegio que no esté establecido en la máscara de acceso. Solo los permisos del propietario del archivo y otros no se ven afectados por la máscara de acceso.

En la línea de comando, las herramientas `getfacl` y `setfacl` se pueden usar para leer y modificar las entradas POSIX ACL, respectivamente:

```
user@lab:~$ touch /tmp/mytest
user@lab:~$ getfacl /tmp/mytest
getfacl: Removing leading '/' from absolute path names
# file: tmp/mytest
# owner: user
# group: user
user::rw-
group::r--
other::r--
```

Este ejemplo también muestra que el conjunto de permisos estándar se refleja en la POSIX ACL. En consecuencia, si las POSIX ACL son compatibles en un sistema de archivos, entonces el conjunto de permisos completo está contenido en POSIX ACL.

Revocamos el acceso de lectura a otros usuarios y agregamos acceso de lectura / escritura a `games` de usuario, como se muestra aquí:

```
user@lab:~$ setfacl -m o::0 -m u:games:rw /tmp/mytest
user@lab:~$ getfacl /tmp/mytest
getfacl: Removing leading '/' from absolute path names
# file: tmp/mytest
# owner: user
# group: user
user::rw-
user:games:rw-
group::r--
mask::rw-
```

```
other::---
user@lab:~$ ls -l /tmp/mytest
-rw-rw----+ 1 user user 0 May 16 16:59 /tmp/mytest
```

El parámetro `-m o::0` elimina todos los privilegios de otros usuarios mientras `-m u:games:rw` otorga acceso de lectura/escritura a `games` de usuario. La llamada subsiguiente a `getfacls` muestra la entrada adicional para `user:games` y la entrada modificada para `other`. Además, una entrada `mask` se crea automáticamente para limitar el acceso de todos los grupos y usuarios listados (excepto el propietario del archivo) para leer / escribir.

El resultado del comando `ls` muestra un signo más `+` para indicar la existencia de las entradas ACL adicionales. Como también lo indica el resultado de ls, las herramientas que solo evalúan los bits de modo de archivo no son conscientes de los permisos adicionales, por ejemplo, los privilegios de acceso adicionales para los juegos de usuario no aparecen en la salida estándar de ls u otro aplicaciones de administración de archivos.



Las herramientas forenses que no buscan e interpretan las entradas de POSIX ACL pueden perder los derechos de acceso adicionales que introducen las entradas de ACL. En consecuencia, el investigador puede obtener una falsa impresión de privilegios estrictos y efectivos.

Afortunadamente, la biblioteca Python **pylibacl** se puede usar para leer y evaluar POSIX ACL y, por lo tanto, evitar esa trampa. La biblioteca introduce el módulo `posix1e`, es decir, una referencia al borrador inicial que menciona primero las ACL POSIX. La documentación detallada sobre esta biblioteca está disponible en http://pylibacl.k1024.org/.

El siguiente script es un ejemplo de cómo buscar archivos con las entradas de ACL adicionales:

```
#!/usr/bin/env python

import os
from os.path import join
import posix1e
import re
import stat
import sys

def acls_from_file(filename, include_standard = False):
    """Returns the extended ACL entries from the given
       file as list of the text representation.

       Arguments:
       filename -- the file name to get the ACLs from
```

```
        include_standard -- if True, ACL entries representing
                            standard Linux permissions will be
                            included"""
    result = []
    try:
        acl = posix1e.ACL(file=filename)
    except:
        print 'Error getting ACLs from %s' % filename
        return []

    text = acl.to_any_text(options=posix1e.TEXT_ABBREVIATE | posix1e.
TEXT_NUMERIC_IDS)

    for entry in text.split("\n"):
        if not include_standard and \
           re.search(r'^[ugo]::', entry) != None:
            continue
        result.append(entry)

    return result


def get_acl_list(basepath, include_standard = False):
    """Collects all POSIX ACL entries of a directory tree.

    Arguments:
    basepath -- directory to start from
    include_standard -- if True, ACL entries representing
                        standard Linux permissions will be
                        included"""
    result = {}

    for root, dirs, files in os.walk(basepath):
        for f in dirs + files:
            fullname = join(root, f)

            # skip symbolic links (target ACL applies)
            if stat.S_ISLNK(os.lstat(fullname).st_mode):
                continue

            acls = acls_from_file(fullname, include_standard)
            if len(acls) > 0:
```

```
                result[fullname] = acls

    return result

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print 'Usage %s root_directory' % sys.argv[0]
        sys.exit(1)

    acl_list = get_acl_list(sys.argv[1], False)

    for filename, acls in acl_list.iteritems():
        print "%s: %s" % (filename, ','.join(acls))
```

La clase `posix1e.ACL` representa todos los permisos establecidos en un objeto específico en el sistema de archivos. Cuando se invoca a su constructor con un nombre de archivo como parámetro `file`, representa la ACL de ese archivo. En la función `acls_from_file()`, se utiliza una expresión regular para detectar y, opcionalmente, filtrar los permisos estándar de la representación de texto del conjunto de ACL.

La función `os.walk()` se utiliza para iterar sobre un subárbol del sistema de archivos. Si anda sobre `os.walk()` como en el ejemplo, obtienes un triple en cada iteración que denota lo siguiente:

- El directorio visitado actualmente
- Una lista con todos sus subdirectorios (en relación con el directorio visitado actualmente)
- Una lista con todas sus entradas no direccionales, por ejemplo, archivos y enlaces blandos (en relación con el directorio visitado actualmente)

La comprobación en la última línea resaltada del script es un ejemplo de la información del tipo de archivo de evaluación como se describe en la sección anterior. Se usa para detectar y omitir enlaces simbólicos. Los enlaces simbólicos siempre usan ACL de destino y, en consecuencia, las ACL POSIX en enlaces simbólicos no son compatibles.

Cuando se invoca con `/tmp` como parámetro en nuestra máquina de laboratorio, genera la siguiente salida:

```
/tmp/mytest: u:5:rw-,m::rw-
```

Este resultado muestra que el script detectó las sobras de nuestras primeras pruebas con ACL POSIX: un permiso de lectura/escritura adicional para el usuario (u) ID 5 (es decir, `games` de usuario en la máquina de laboratorio) y una entrada de máscara (m) que limita los privilegios efectivos para leer/escribir. La secuencia de comandos genera los ID de usuario numéricos porque pylibacl usaría de otra manera su /etc/passwd de la estación de trabajo para buscar los nombres de usuario.

Si ejecuta este script en una copia del sistema de archivos que contiene su evidencia, listará cada objeto del sistema de archivos con permisos más allá del conjunto de permisos estándar de Linux.



La mayoría de los sistemas Linux estándar y sus aplicaciones no usan POSIX ACL. Por lo tanto, si encuentra entradas adicionales de ACL POSIX durante su investigación, es una buena idea verificar minuciosamente si estas ACL POSIX fueron el resultado de una operación del sistema legítima y benigna.

Lectura de capacidades de archivos con Python

Tradicionalmente, en Linux, existen dos tipos de privilegios administrativos: root y no root. Si se concede un proceso a los privilegios de root, es decir, se ejecuta con UID 0, entonces puede pasar por alto todas las restricciones de seguridad del núcleo de Linux. Por otro lado, si un proceso no se ejecuta con estos privilegios de raíz, se aplican todas las restricciones de seguridad del núcleo.

Para reemplazar este mecanismo de **todo o nada** con un sistema más preciso, se presentaron las **capacidades de Linux**. La página de manual correspondiente lo describe como el siguiente:

Con el fin de realizar comprobaciones de permisos, las implementaciones tradicionales de UNIX distinguen dos categorías de procesos: procesos privilegiados (cuyo ID de usuario efectivo es 0, denominado superusuario o root) y procesos sin privilegios (cuyo UID efectivo es distinto de cero).

Los procesos privilegiados omiten todas las comprobaciones de permisos del kernel, mientras que los procesos sin privilegios están sujetos a la verificación de permisos completa en función de las credenciales del proceso (generalmente: UID efectiva, GID efectiva y lista de grupos suplementarios).

Comenzando con el kernel 2.2, Linux divide los privilegios tradicionalmente asociados con superusuario en unidades distintas, conocidas como capacidades, que pueden habilitarse e inhabilitarse independientemente. Las capacidades son un atributo por hilo.

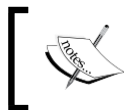


¿Qué capacidades existen?

La lista de capacidades de Linux se puede encontrar en el archivo /usr/include/linux/capability.h en un sistema Linux estándar. En la página de manual de capacidades se proporciona una forma más legible para el ser humano. Se puede ver a través de `man 7 capabilities`. Las capacidades de Linux incluyen todos los permisos especiales concedidos al usuario root, por ejemplo, anular los permisos de los archivos, usar conexiones de red sin procesar, etc.

Las capacidades se pueden asignar a los subprocesos de los procesos durante la ejecución y a los ejecutables en el sistema de archivos. En cualquier caso, siempre hay tres conjuntos de capacidades:

- **permitted set** (**p**): El permitted set contiene todas las capacidades que un hilo puede solicitar. Si se inicia un ejecutable, su conjunto permitido se usa para inicializar el conjunto de procesos permitido.
- **inheritable set** (**i**): El inheritable set de un conjunto de ejecución define las capacidades que se pueden reenviar desde el subproceso a un proceso secundario. Sin embargo, solo las capacidades definidas en el conjunto heredable del archivo ejecutable del proceso secundario se reenvían al proceso secundario. Por lo tanto, una capacidad solo se hereda si está en el conjunto heredable del proceso principal y en el atributo de archivo del ejecutable secundario.
- **effective set** (**e**): Este es el conjunto de capacidades que el kernel de Linux realmente comprueba cuando se solicita una operación privilegiada de un subproceso de ejecución. Al llamar a `cap_set_proc()`, un proceso puede deshabilitar o habilitar las capacidades. Solo las capacidades en el conjunto permitido (p) pueden estar habilitadas. En el sistema de archivos, el conjunto efectivo está representado por un solo bit. Si se establece este bit, el ejecutable se inicia con todas las capacidades permitidas que también son efectivas. Si el bit no está configurado, el nuevo proceso comienza sin las capacidades efectivas.



Las capacidades conceden privilegios administrativos a los ejecutables sin requerir el bit SUID en el modo de archivo. Por lo tanto, durante una investigación forense, todas las capacidades de archivo deben estar documentadas.

Al usar Python ctypes, la biblioteca compartida `libcap.so.2` puede utilizarse para recuperar todas las capacidades de archivo de un árbol de directorio, de la siguiente manera:

```
#!/usr/bin/env python

import ctypes
import os
from os.path import join
import sys

# load shared library
libcap2 = ctypes.cdll.LoadLibrary('libcap.so.2')

class cap2_smart_char_p(ctypes.c_char_p):
    """Implements a smart pointer to a string allocated
       by libcap2.so.2"""
    def __del__(self):
```

```
        libcap2.cap_free(self)

# note to ctypes: cap_to_text() returns a pointer
# that needs automatic deallocation
libcap2.cap_to_text.restype = cap2_smart_char_p

def caps_from_file(filename):
    """Returns the capabilities of the given file as text"""

    cap_t = libcap2.cap_get_file(filename)
    if cap_t == 0:
        return ''
    return libcap2.cap_to_text(cap_t, None).value


def get_caps_list(basepath):
    """Collects file capabilities of a directory tree.

    Arguments:
    basepath -- directory to start from"""

    result = {}
    for root, dirs, files in os.walk(basepath):
        for f in files:
            fullname = join(root, f)
            caps = caps_from_file(fullname)
            if caps != '':
                result[fullname] = caps

    return result

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print 'Usage %s root_directory' % sys.argv[0]
        sys.exit(1)

    capabilities = get_caps_list(sys.argv[1])

    for filename, caps in capabilities.iteritems():
        print "%s: %s" % (filename, caps)
```

La primera línea resaltada carga la biblioteca `libcap.so.2` para uso directo en Python. Como la memoria para la representación de texto de las capacidades se asigna en esta biblioteca, es responsabilidad de la persona que llama, es decir, nuestro script, desasignar esta memoria después de su uso. La solución para esta tarea, que se eligió aquí, es extender la representación predeterminada `ctype` de **puntero a carácter**, es decir, `ctype.c_char_p`. La clase `cap2_smart_char_p` resultante es una versión simple del llamado puntero inteligente: si la representación de Python de los objetos de esta clase se está destruyendo, los objetos llamarán automáticamente a `cap_free()` para liberar los recursos correspondientes que fueron asignados previamente por `libcap.so.2`

Con la función de biblioteca `cap_get_file()`, se pueden recuperar las capacidades de un archivo. La llamada subsiguiente a `cap_to_text ()` transforma esta representación interna en texto legible por humanos.

Si el script se guarda en `chap03_capabilities.py`, entonces se puede invocar en la máquina de laboratorio como se muestra a continuación:

```
user@lab:~$ python chap03_capabilities.py /usr
```

Por supuesto, el resultado es altamente dependiente de la versión y distribución de Linux. Puede ser similar al siguiente:

```
/usr/bin/gnome-keyring-daemon: = cap_ipc_lock+ep
```

Este resultado significa que solo un ejecutable en `/usr` tiene las capacidades especiales establecidas: `/usr/bin/gnome-keyring-daemon`. El nombre de la capacidad viene dado por la constante `cap_ipc_lock`, esta capacidad está en el conjunto permitido y es efectiva inmediatamente al iniciar este programa como se denota por `+ ep`.

Para resolver el significado de `cap_ipc_lock`, llamaremos a lo siguiente:

```
user@lab:~$ man 7 capabilities
```

Luego buscaremos CAP_IPC_LOCK. Esto revela que la capacidad otorga el derecho de bloquear las partes o la totalidad de la memoria de un proceso en la RAM y evitar el intercambio de ese proceso. Como `gnome-keyring-daemon` almacena credenciales de usuario en RAM, tener el privilegio de evitar que estas credenciales se escriban en el intercambio es muy recomendable desde una perspectiva de seguridad.



Actualmente, la mayoría de las distribuciones de Linux estándar hacen poco uso de la función de capacidad de archivo. Por lo tanto, las capacidades del archivo descubierto, especialmente aquellas que no son necesarias para el funcionamiento normal, pueden ser el primer indicador de la manipulación del sistema.

información del archivo de clustering

En la sección anterior, le mostramos cómo recuperar y recopilar metadatos de archivos del sistema de archivos Linux / Unix. En esta sección, proporcionaremos ejemplos para ubicar los cambios en los metadatos del sistema de archivos, que pueden ser interesantes para una mayor inspección por parte del investigador.

Creación de histogramas

La creación de histogramas es el proceso de agrupar los datos en contenedores de igual tamaño y dibujar el tamaño de estos contenedores. Con Python, el trazado de estos histogramas se puede lograr fácilmente utilizando el módulo Python **matplotlib**. Una documentación detallada que incluye los casos de uso, ejemplos y código fuente de Python está disponible en http://matplotlib.org/.

La siguiente secuencia de comandos de Python se puede usar para generar y mostrar los histogramas de los tiempos de acceso a archivos y los tiempos de modificación de archivos de un árbol de directorios:

```
#!/usr/bin/env python

from datetime import datetime
from matplotlib.dates import DateFormatter
import matplotlib.pyplot as plt
import os
from os.path import join
import sys

# max. number of bars on the histogram
NUM_BINS = 200

def gen_filestats(basepath):
    """Collects metadata about a directory tree.

    Arguments:
    basepath -- root directory to start from

    Returns:
    Tuple with list of file names and list of
    stat results."""

    filenames = []
    filestats = []

    for root, dirs, files in os.walk(basepath):
        for f in files:
```

```
            fullname = join(root, f)
            filenames.append(fullname)
            filestats.append(os.lstat(fullname))
    return (filenames, filestats)

def show_date_histogram(times, heading='', block=False):
    """Draws and displays a histogram over the given timestamps.

    Arguments:
    times -- array of time stamps as seconds since 1970-01-01
    heading -- heading to write to the drawing
    block --- if True, the graph window waits for user interaction"""

    fig, ax = plt.subplots()

    times = map(lambda x: datetime.fromtimestamp(x).toordinal(),
times)

    ax.hist(times, NUM_BINS)
    plt.xlabel('Date')
    plt.ylabel('# of files')
    plt.title(heading)

    ax.autoscale_view()

    ax.xaxis.set_major_formatter(DateFormatter('%Y-%m-%d'))
    fig.autofmt_xdate()

    fig.show()
    if block:
        plt.show()

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print 'Usage %s base_directory' % sys.argv[0]
        sys.exit(1)

    path = sys.argv[1]

    (names, stats) = gen_filestats(path)

    # extract time stamps
    mtimes = map(lambda x: x.st_mtime, stats)
```

```
    atimes = map(lambda x: x.st_atime, stats)

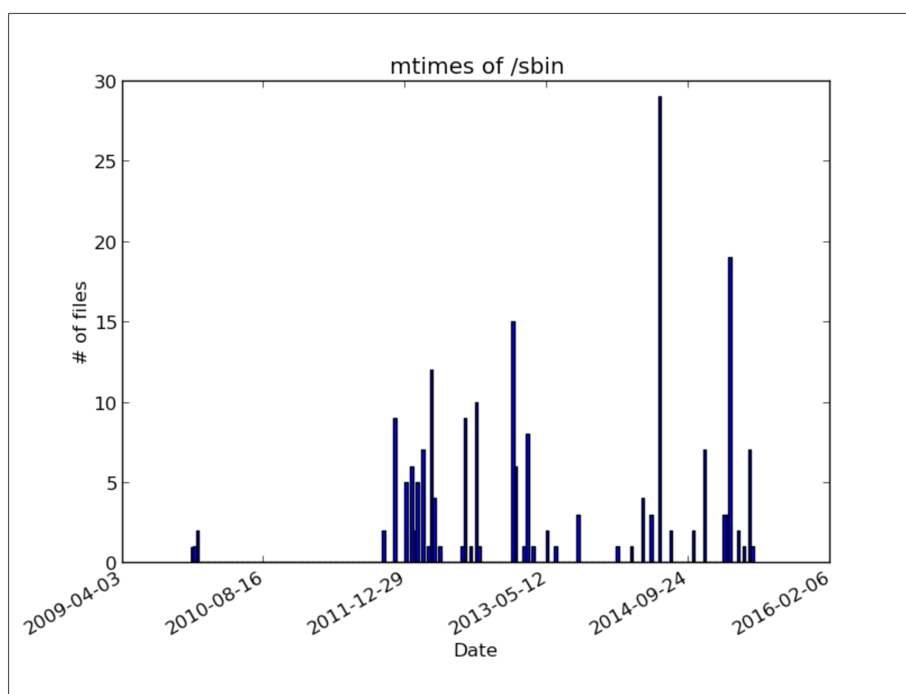
    show_date_histogram(mtimes, 'mtimes of ' + path)
    show_date_histogram(atimes, 'atimes of ' + path, True)
```

La función `gen_filestats()` itera el árbol de directorios y recopila todos los datos de inodo. La función `show_date_histogram()` se usa para generar y mostrar los datos como un histograma.

En la primera línea resaltada del código, se cambia la codificación de la marca de tiempo. Esto es necesario porque los datos de inodo nos dan las marcas de tiempo como el número de segundos desde 1970-01-01. Este formato es lo que espera `datetime.fromtimestamp()`. Sin embargo, Matplotlib necesita marcas de tiempo en el número de días desde el 0001-01-01 del calendario gregoriano. Afortunadamente, la clase `datetime` puede proporcionar esta representación con su método `toordinal()`.

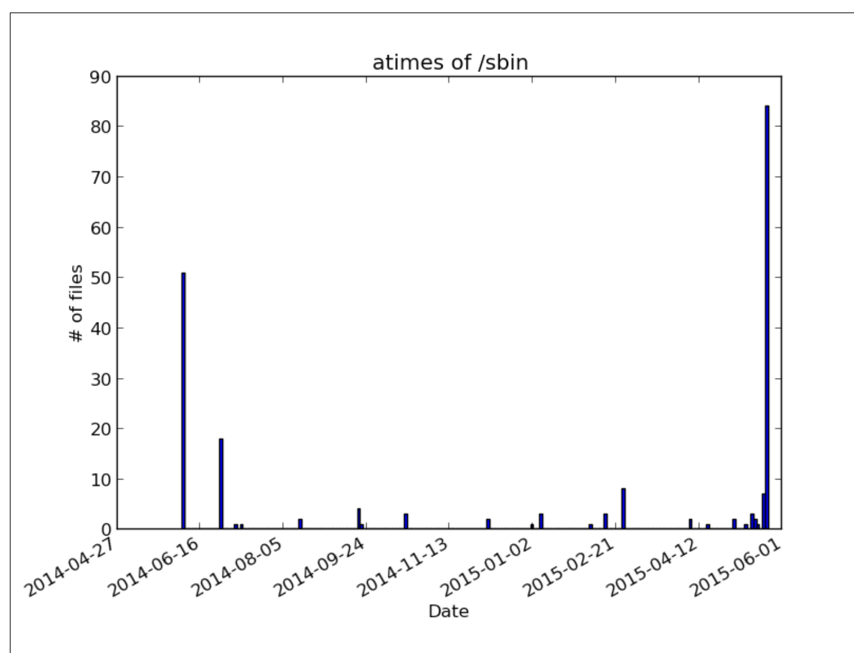
La siguiente línea resaltada es la generación real y el dibujo del histograma en la siguiente figura. Todas las otras declaraciones de `show_date_histogram ()` simplemente sirven para añadir etiquetas y formato al dibujo.

El siguiente es un resultado de muestra del directorio `/sbin` en un sistema de escritorio Linux estándar:



Aquí, las fechas de las principales actualizaciones del sistema son claramente visibles. Un investigador debe saber que los metadatos del archivo y estos histogramas no contienen información histórica del archivo. Por lo tanto, a partir del histograma anterior, no se puede deducir que hubo pocas o ninguna actualización de seguridad antes de diciembre de 2011. Es más probable que la mayoría de los archivos que fueron parchados antes de diciembre de 2011 se hayan modificado más adelante, por lo tanto, enmascarando parches más antiguos en el histograma

Echemos un vistazo a la distribución del tiempo de acceso de este directorio:



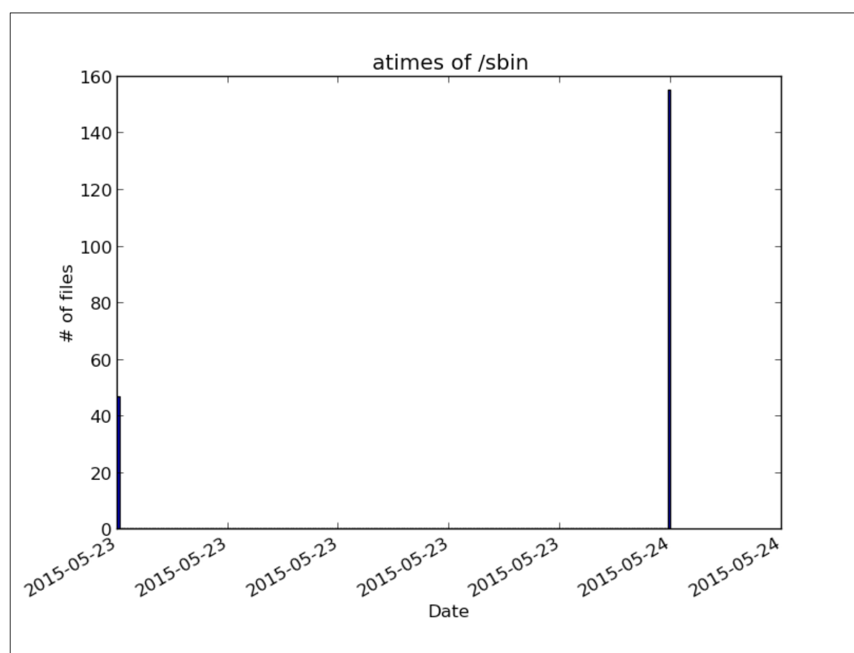
Este histograma proporciona información sobre el patrón de acceso de este directorio. En primer lugar, el rastreo de marcas de hora en tiempo real está habilitado en los sistemas. De lo contrario, las marcas de tiempo de acceso actual no serían visibles en el histograma. Alrededor de la mitad de los archivos se han leído recientemente. Esta información puede usarse para verificar la información sobre el momento en que se adquirió la evidencia o cuando el operador del sistema afirmó haber desconectado el sistema.

Además, es muy probable que los contenidos de este directorio no se escanearon regularmente en busca de virus y no se incluyeron recientemente en un archivo. Ambas acciones generalmente actualizan la marca de tiempo de Atime.

Si se emite el siguiente comando en el sistema, entonces `/sbin` escanea en busca de virus. Por supuesto, el escáner debe leer cada archivo en ese directorio para escanear su contenido:

```
user@lab:~$ clamscan –i /sbin
```

El diagrama de atime de `/sbin` refleja los cambios, de la siguiente manera:



Los cambios son obvios: la mayoría de las barras se han colapsado en una en el momento actual, es decir, en el momento del análisis de virus. La escala de tiempo se extiende a un solo día. En consecuencia, la barra de la izquierda también se puede considerar como un resultado de la exploración de virus.



Si hay un directorio que tiene todas las marcas de tiempo `atime` en una sola fecha, este directorio fue copiado recientemente, escaneado en busca de virus o empaquetado en un archivo. Por supuesto, con suficientes derechos de acceso, las marcas de tiempo también podrían haberse configurado manualmente.

Técnicas avanzadas de histograma

En la sección anterior, los histogramas se utilizaron para conocer los metadatos del sistema de archivos. Sin embargo, estos histogramas tienen varias desventajas, como sigue:

- Todas las barras de histograma tienen el mismo ancho
- Las barras no se colocan según el agrupamiento real de los datos, por ejemplo, un agrupamiento puede distribuirse en dos barras
- Los valores atípicos desaparecen, es decir, las barras bajas se confunden fácilmente con las barras vacías

Por lo tanto, esta sección presenta un ejemplo de cómo usar **algoritmos simples de aprendizaje automático** para una agrupación más inteligente de los datos. Una biblioteca de aprendizaje de máquina ampliamente utilizada para Python es **scikit-learn.** Entre otros dominios, proporciona varios algoritmos para agrupar los datos de entrada. Recomendamos visitar http://scikit-learn.org para obtener una visión general de todos los algoritmos y ejemplos de su uso. La siguiente secuencia de comandos de Python utiliza el algoritmo `DBSCAN` de scikit-learn para generar clústeres de un ancho determinado (en días):

```
#!/usr/bin/python

from datetime import date
import numpy as np
import os
from os.path import join
from sklearn.cluster import DBSCAN
import sys

def gen_filestats(basepath):
    """Collects metadata about a directory tree.

    Arguments:
    basepath -- root directory to start from

    Returns:
    Tuple with list of file names and list of
    stat results."""

    filenames = []
    filestats = []

    for root, dirs, files in os.walk(basepath):
        for f in files:
```

```
            fullname = join(root, f)
            filenames.append(fullname)
            filestats.append(os.lstat(fullname))
    return (filenames, filestats)


def _calc_clusters(data, eps, minsamples):
    samples = np.array(data)
    db = DBSCAN(eps=eps, min_samples=minsamples).fit(samples)
    return (db.labels_, db.core_sample_indices_)

def calc_atime_clusters(stats, days=1, mincluster=5):
    """Clusters files regarding to their 'last access' date.

    Arguments:
    stats -- file metadata as returned as 2nd element by gen_filestats
    days -- approx. size of a cluster (default: accessed on same day)
    mincluster -- min. number of files to make a new cluster

    Returns:
    Tuple with array denoting cluster membership
    and indexes of representatives of cluster cores"""

    atimes = map(lambda x: [x.st_atime], stats)
    return _calc_clusters(atimes, days * 24 * 3600, mincluster)

def calc_mtime_clusters(stats, days=1, mincluster=5):
    """Clusters files regarding to their 'last modified' date.

    Arguments:
    stats -- file metadata as returned as 2nd element by gen_filestats
    days -- approx. size of a cluster (default: accessed on same day)
    mincluster -- min. number of files to make a new cluster

    Returns:
    Tuple with array denoting cluster membership
    and indexes of representatives of cluster cores"""

    mtimes = map(lambda x: [x.st_mtime], stats)
    return _calc_clusters(mtimes, days * 24 * 3600, mincluster)


def calc_histogram(labels, core_indexes, timestamps):
```

```
    # reserve space for outliers (label -1), even if there are none
    num_entries = len(set(labels)) if -1 in labels else
len(set(labels))+1

    counters = [0] * num_entries
    coredates = [0] * num_entries

    for c in core_indexes:
        i = int(c)
        coredates[int(labels[i])+1] = timestamps[i]

    for l in labels:
        counters[int(l)+1] += 1

    return zip(coredates, counters)

def print_histogram(histogram):
    # sort histogram by core time stamps
    sort_histo = sorted(histogram, cmp=lambda x,y: cmp(x[0],y[0]))

    print '[date around] [number of files]'
    for h in sort_histo:
        if h[0] == 0:
            print '<outliers>',
        else:
            t = date.fromtimestamp(h[0]).isoformat()
            print t,
        print '    %6d' % h[1]



if   name__ == '  main  ':
    if len(sys.argv) < 2:
        print 'Usage %s base_directory [number of days in one
cluster]' % sys.argv[0]
        sys.exit(1)

    days = 1
    if len(sys.argv) > 2:
        days = int(sys.argv[2])

    names, stats = gen_filestats(sys.argv[1])

    print '%d files to analyze...' % len(names)
```

```
        atime_labels, atime_cores = calc_atime_clusters(stats, days)
        mtime_labels, mtime_cores = calc_mtime_clusters(stats, days)

        atimes = map(lambda x: x.st_atime, stats)
        mtimes = map(lambda x: x.st_mtime, stats)

        ahisto = calc_histogram(atime_labels, atime_cores, atimes)
        mhisto = calc_histogram(mtime_labels, mtime_cores, mtimes)

        print "\n=== Access time histogram ==="
        print_histogram(ahisto)

        print "\n=== Modification time histogram ==="
        print_histogram(mhisto)
```

La función `gen_filestats()` es idéntica a la versión utilizada para los histogramas básicos en la sección anterior. Las funciones `calc_atime_clusters()` y `calc_mtime_ clusters()` extraen el tiempo de acceso y modificación de la información recopilada y la transmiten a la generación de clúster en `_calc_clusters`. El `DBSCAN` se inicializa con dos parámetros: el tamaño de un clúster (eps, en segundos) y el número mínimo de datos de muestra que puede hacer un clúster (`min_samples`). Una vez que se establecen los parámetros del algoritmo, los datos se introducen con el fin de agruparlos mediante el método `fit ()`.

El resultado de esta agrupación es una tupla que consta de **etiquetas** y una lista de índices por etiqueta. Una etiqueta se correlaciona con un clúster que se encuentra en los datos de entrada. Su valor es el centro, es decir, la fecha promedio, de todas las fechas del clúster. La etiqueta especial `-1` actúa como un contenedor para todos los valores atípicos, es decir, todos los datos que no se podrían asignar a un clúster.

La función `calc_histogram()` cuenta el tamaño de cada clúster y devuelve el histograma, es decir, las etiquetas y el número de entradas como matriz bidimensional.

Podemos ejecutar este script de Python en el directorio /sbin, de la siguiente manera:

```
user@lab:~$ python timecluster.py /sbin
```

El resultado puede ser similar al siguiente:

```
202 files to analyze...


=== Access time histogram ===
```

```
[date around] [number of files]
<outliers>          0
2015-05-24        202

=== Modification time histogram ===
[date around] [number of files]
<outliers>         64
2011-11-20          9
2012-02-09          5
2012-03-02          6
2012-03-31         11
2012-07-26          6
2012-09-11         10
2013-01-18         15
2013-01-26          6
2013-03-07          8
2014-06-18         29
2014-11-20          7
2015-02-16         19
2015-05-01          7
```

Aquí, el histograma de tiempo de acceso muestra solo una entrada, lo que refleja nuestro análisis anterior del directorio. Además, todas las principales actualizaciones del sistema en el pasado reciente se muestran en el histograma del tiempo de modificación.

Con esta herramienta, el investigador puede agrupar la información del sistema de archivos para detectar el escaneo o la extracción de los directorios, así como los parches de seguridad descuidados. Además, el grupo especial -1 se puede analizar para obtener los nombres de los archivos, que se modificaron fuera de las principales actualizaciones del sistema.

Resumen

En este capítulo, vimos ejemplos prominentes de las propiedades especiales de los sistemas de Microsoft Windows y Linux (y similar a Linux). Ahora puede extraer información del registro de eventos de Windows, el registro de Windows, los archivos de Linux y el sistema de archivos de Linux. Usando Python, toda esta información puede ser analizada automática y semiautomáticamente para los Indicadores de Compromiso, reconstruyendo la actividad reciente del sistema y los signos de exfiltración.

Además, leer las capacidades del sistema de archivos nos muestra cómo usar ctype para cargar las bibliotecas nativas para ayudar al análisis del sistema de archivos.

En la agrupación de información de archivos, proporcionamos el primer ejemplo sobre cómo usar los algoritmos básicos de aprendizaje automático para soportar el análisis forense.

Ahora que echamos un vistazo a los sistemas locales, iremos al siguiente capítulo y echaremos un vistazo al tráfico de red y cómo buscar los **Indicadores de Compromiso (IOC)** allí.

4

Uso de Python para análisis forense de redes

En este capítulo, nos centraremos en las partes de la investigación forense que son específicas de la capa de red. Elegiremos uno de los paquetes de Python más utilizados para manipular y analizar el tráfico de red (**Scapy**), así como un framework de código abierto recientemente lanzado por el Laboratorio de Investigación del Ejército de los EE. UU. (**Dshell**). Para ambos conjuntos de herramientas, hemos seleccionado los ejemplos de evidencia interesante. Este capítulo te enseñará lo siguiente:

- Cómo buscar el IOC en el tráfico de red
- Cómo extraer archivos para un análisis más detallado
- Cómo monitorear los archivos accedidos a través del **Bloque de mensajes del servidor (SMB)**
- Cómo construir su propio escáner de puertos

Usando Dshell durante una investigación

Dshell es un juego de herramientas de análisis forense de red basado en Python que es desarrollado por el Laboratorio de Investigación del Ejército de Estados Unidos y lanzado como código abierto a finales de 2014. Puede ayudar a hacer las investigaciones forenses en la capa de red un poco más fácil. El juego de herramientas viene con una gran cantidad de descodificadores que se pueden usar fuera de la caja y son muy útiles. Algunos de estos decodificadores son los siguientes:

- **dns**: Extrae y resume consultas/respuestas de DNS
- **reservedips**: Identifica las resoluciones de DNS que caen en el espacio de IP reservado
- **large-flows**: Muestra los flujos netos que al menos han transferido 1 MB

- **rip-http**: Extrae los archivos del tráfico HTTP
- **protocols**: Identifica los protocolos no estándar
- **synrst**: Detecta los intentos fallidos de conectarse (SYN seguido de un RST/ACK)

Dshell puede instalarse en nuestro entorno de laboratorio clonando las fuentes de GitHub en https://github.com/USArmyResearchLab/Dshell y ejecutando `install-ubuntu. py` Este script descargará automáticamente los paquetes faltantes y creará los ejecutables que necesitaremos luego. Dshell se puede usar contra los archivos pcap que se han grabado durante los incidentes o como resultado de una alerta de IDS. Un archivo de **captura de paquetes (pcap)** es creado por libpcap (en Linux) o WinPcap (en Windows).

En la siguiente sección, explicaremos cómo un investigador puede usar Dshell demostrando el conjunto de herramientas con escenarios del mundo real que se recopilan de http://malware-traffic-analysis.net.

El primer ejemplo es un archivo ZIP malicioso que un usuario ha encontrado a través de un enlace de correo electrónico. El usuario inició sesión en Gmail y presionó el enlace de descarga en el correo. Esto se puede ver fácilmente con el decodificador web de Dshell, de la siguiente manera:

```
user@lab:~$ source labenv/bin/activate
(labenv)user@lab:~$ ./dshell

(labenv)user@lab:~$ Dshell> decode -d web infected_email.pcap

web 2015-05-29 16:23:44     10.3.162.105:62588 ->   74.125.226.181:80
** GET mail.google.com/ HTTP/1.1
// 200 OK  2015-05-29 14:23:40 **

web 2015-05-29 16:24:15     10.3.162.105:62612 <-    149.3.144.218:80
** GET sciclubtermeeuganee.it/wp-content/plugins/feedweb_data/pdf_efax_
message_3537462.zip HTTP/1.1
// 200 OK  2015-05-28 14:00:22 **
```

Al mirar el extracto de tráfico anterior, el archivo ZIP podría ser el primer indicador de compromiso. Por lo tanto, deberíamos analizarlo más a fondo. La forma más fácil de hacerlo es extraer el archivo ZIP del archivo pcap y comparar su hash md5 con la base de datos VirusTotal:

```
(labenv)user@lab:~$ Dshell> decode -d rip-http --bpf "tcp and port
62612" infected_email.pcap


rip-http 2015-05-29 16:24:15     10.3.162.105:62612 <-
149.3.144.218:80    ** New file: pdf_efax_message_3537462.zip
(sciclubtermeeuganee.it/wp-
content/plugins/feedweb_data/pdf_efax_ message_3537462.zip) **
 --> Range: 0 - 132565
rip-http 2015-05-29 16:24:15     10.3.162.105:62612 <-
149.3.144.218:80    ** File done: ./pdf_efax_message_3537462.zip
(sciclubtermeeuganee.it/wp-
content/plugins/feedweb_data/pdf_efax_ message_3537462.zip) **


(labenv)user@lab:~$ Dshell> md5sum
pdf_efax_message_3537462.zip 9cda66cba36af799c564b8b33c390bf4
                                  pdf_efax_message_3537462.zip
```

En este caso simple, nuestra primera suposición fue correcta ya que el archivo ZIP descargado contiene otro ejecutable que parte de un kit de malware infostealer, como se ve en la siguiente captura de pantalla:

virustotal

SHA256: 78d00fd08085eb2c4353474e506305da4bda767d75f3ce4c28b826490e0d1b89
Dateiname: pdf_efax_message_3537462.zip
Erkennungsrate: 40 / 57
Analyse-Datum: 2015-06-11 07:29:15 UTC (vor 4 Tage, 13 Stunden)

2 0

Analyse | File detail | Relationships | Zusätzliche Informationen | Kommentare 0 | Bewertungen

Antivirus	Ergebnis	Aktualisierung
ALYac	Trojan.GenericKD.2447672	20150611
AVG	Generic36.BNIU	20150611
AVware	Lookslike.Win32.Crowti.an!ag (v)	20150611
Ad-Aware	Trojan.GenericKD.2447672	20150611

Otro ejemplo muy bueno es la búsqueda de los archivos a los que accede en un recurso compartido de red a través del protocolo SMB. Esto puede ser muy útil cuando se trata de averiguar si un atacante pudo acceder o incluso exfiltrar los datos y, si tuvo éxito, qué datos se han filtrado potencialmente:

```
(labenv)user@lab:~$ Dshell> decode -d smbfiles exfiltration.pcap


smbfiles 2005-11-19 04:31:58    192.168.114.1:52704 ->
192.168.114.129:445   ** VNET3\administrator \\192.168.114.129\TEST\
torture_qfileinfo.txt (W) **
smbfiles 2005-11-19 04:31:58    192.168.114.1:52704 ->
192.168.114.129:445   ** VNET3\administrator \\192.168.114.129\
TESTTORTUR~1.TXT (-) **
smbfiles 2005-11-19 04:31:58    192.168.114.1:52705 ->
192.168.114.129:445   ** VNET3\administrator \\192.168.114.129\TEST\
testsfileinfo\fname_test_18.txt (W) **
```

Con la ayuda del decodificador **rip-smb-uploads**, Dshell también puede extraer automáticamente todos los archivos cargados del archivo pcap grabado. Otro ejemplo interesante es buscar el IOC con la ayuda de las reglas de snort, que Dshell también puede hacer de la siguiente manera:

```
(labenv)user@lab:~$ Dshell> decode -d snort malicious-word-document.
pcap --snort_rule 'alert tcp any 443 -> any any (msg:"ET
CURRENT_EVENTS Tor2Web .onion Proxy Service SSL Cert (1)";
content:"|55 04 03|"; content:"*.tor2web.";)' –snort_alert


snort 2015-02-03 01:58:26      38.229.70.4:443   --
192.168.120.154:50195 ** ET CURRENT_EVENTS Tor2Web .onion Proxy Service
SSL Cert (1) **
snort 2015-02-03 01:58:29      38.229.70.4:443   --
192.168.120.154:50202 ** ET CURRENT_EVENTS Tor2Web .onion Proxy Service
SSL Cert (1) **
snort 2015-02-03 01:58:32      38.229.70.4:443   --
192.168.120.154 :50204 ** ET CURRENT_EVENTS Tor2Web .onion Proxy
Service SSL Cert (1) **
```

En este ejemplo, abrimos un documento de Word potencialmente malicioso que hemos recibido en un correo electrónico no deseado. El documento de Word intenta descargar el malware **Vawtrak** y, por lo tanto, se comunica a través de la **red Tor**. La regla snort que estamos utilizando proviene de Amenazas emergentes, (consulte http: // www. Emergingthreats.net/), y está buscando certificados SSL conocidos para el servicio **Tor2Web** (un servicio para permitir a los usuarios acceder a los **servicios de Tor Onion** sin usar el Tor Browser). Se pueden hacer verificaciones similares usando todas las reglas de snort disponibles y pueden ser muy útiles si está buscando un ataque específico dentro de la red.

Como alternativa a los archivos pcap mostrados, todos los ejemplos demostrados también se pueden ejecutar en una conexión de red activa con la ayuda del indicador `-i interface_name` como se muestra a continuación:

```
(labenv)user@lab:~$ Dshell> decode -d netflow -i eth0

2015-05-15 21:35:31.843922   192.168.161.131 ->    85.239.127.88  (None
-> None)  TCP   52007      80     0      0        0        0  5.1671s
2015-05-15 21:35:31.815329   192.168.161.131 ->    85.239.127.84  (None
-> None)  TCP   46664      80     0      0        0        0  5.1976s
2015-05-15 21:35:32.026244   192.168.161.131 ->   208.91.198.88  (None
-> None)  TCP   40595      80     9     25     4797   169277  6.5642s
2015-05-15 21:35:33.562660   192.168.161.131 ->   208.91.198.88  (None
-> None)  TCP   40599      80     9     19     4740    85732  5.2030s
2015-05-15 21:35:32.026409   192.168.161.131 ->   208.91.198.88  (None
-> None)  TCP   40596      80     7      8     3843   121616  6.7580s
2015-05-15 21:35:33.559826   192.168.161.131 ->   208.91.198.88  (None
-> None)  TCP   40597      80     5     56     2564   229836  5.2732s
```

En este ejemplo, estamos generando los datos de flujo neto de una conexión activa. Dshell está escrito puramente en Python, lo que lo hace altamente adaptable a todas las necesidades de los investigadores forenses y también se puede usar en una cadena con otras herramientas o procesos predefinidos.

Si quiere probar esto, puede descargar los archivos de muestra de `http://www.emergingthreats.net/`.

Uso de Scapy durante una investigación

Otra gran herramienta basada en Python para analizar y manipular el tráfico de red es **Scapy**. Según el sitio web del desarrollador, http://www.secdev.org/projects/scapy/:

> "*Scapy es un poderoso programa interactivo de manipulación de paquetes. Es capaz de forjar o descifrar paquetes de una gran cantidad de protocolos, enviarlos por cable, capturarlos, hacer coincidir solicitudes y respuestas, y mucho más*".

Scapy difiere de las herramientas estándar (y también de Dshell) al proporcionar a un investigador la capacidad de escribir pequeños scripts de Python que pueden manipular o analizar el tráfico de la red, ya sea en forma grabada o en tiempo real. Además, Scapy tiene la capacidad de realizar disecciones profundas de paquetes, huellas dactilares pasivas OS o trazar a través de herramientas de terceros, como **GnuPlot**, ya que las características integradas ya están disponibles.

El siguiente script de Python, que se extrae de *Desarrolle sus propias herramientas forenses: Una taxonomía de las bibliotecas de Python útiles para el análisis forense, la sala de lectura de SANS Institute InfoSec*, es un breve ejemplo de lo poderoso que es Scapy:

```
import scapy, GeoIP
from scapy import *

geoIp = GeoIP.new(GeoIP.GEOIP_MEMORY_CACHE)
def locatePackage(pkg):
  src=pkg.getlayer(IP).src
  dst=pkg.getlayer(IP).dst
  srcCountry = geoIp.country_code_by_addr(src)
  dstCountry = geoIp.country_code_by_addr(dst)
  print src+"("+srcCountry+") >>
"+dst+"("+dstCountry+")\n" try:
  while True:
    sniff(filter="ip", prn=locatePackage, store=0)
except KeyboardInterrupt:
  print "\n" + "Scan Aborted!"
```

Este script registra las estadísticas sobre la geolocalización de la fuente de la dirección IP y el destino de una conexión de red en curso. Después de importar el paquete Scapy en nuestro script de Python, llamamos a la función de sniff y usamos un filtro para detectar solo los paquetes IP. El último parámetro en la función sniff es muy importante si planea ejecutar secuencias de comandos Scapy durante mucho tiempo. Con la ayuda del parámetro store, puede decirle a Scapy que no almacene en caché todos los paquetes en la memoria RAM durante el tiempo de ejecución y que, por lo tanto, haga que el script sea más rápido y ahorre recursos. La función posterior busca la geolocalización de la dirección IP de origen y de destino que se extrae de cada paquete.

En el siguiente ejemplo, ilustraremos cómo construir un escáner de puertos muy simple con la ayuda de Scapy, de la siguiente manera:

```
#!/usr/bin/env python

import sys
from scapy.all import *

targetRange = sys.argv[1]
targetPort = sys.argv[2]
conf.verb=0

p=IP(dst=targetRange)/TCP(dport=int(targetPort),
flags="S") ans,unans=sr(p, timeout=9)

for answers in ans:
        if answers[1].flags == 2:
                print answers[1].src
```

Este pequeño script puede escanear rangos IP completos para un puerto abierto dado. Si está buscando los servidores web que están escuchando en el puerto 80, puede usar el script, de la siguiente manera:

```
(labenv)user@lab:~$ ./scanner.py 192.168.161.1/24 80
WARNING: No route found for IPv6 destination :: (no default route?)
Begin emission:..........
192.168.161.12
192.168.161.34
192.168.161.111
....
```

También podemos usar el **Protocolo de resolución de direcciones (ARP)** para un reconocimiento de todo el rango de red al que está conectado nuestro sistema. Con la ayuda del siguiente script, obtenemos una tabla bien impresa con todas las direcciones IP que están en línea y también sus direcciones MAC correspondientes:

```
#! /usr/bin/env python

import sys
from scapy.all import srp,Ether,ARP,conf

if len(sys.argv) != 2:
        print "Usage: arp_ping <net> (e.g.,: arp_ping
        192.168.1.0/24)" sys.exit(1)

conf.verb=0
ans,unans=srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=sys.argv[1]),
timeout=9)

print r"+------------------+-----------------+"
print r"|       MAC        |        IP       |"
print r"+------------------+-----------------+"
for snd,rcv in ans:
        print rcv.sprintf(r" %Ether.src% | %ARP.psrc%")
print r"+------------------+-----------------+"
```

Al ejecutar el script, recibiremos algo similar a esto:

```
(labenv)user@lab:~$ ./arp_ping.py 192.168.161.131/24
WARNING: No route found for IPv6 destination :: (no default route?)
```

```
+------------------+-----------------+
|       MAC        |        IP       |
+------------------+-----------------+
 00:50:56:c0:00:08 | 192.168.161.1
 00:50:56:f5:d3:83 | 192.168.161.2
 00:50:56:f1:2d:28 | 192.168.161.254
+------------------+-----------------+
```

Los scripts como estos dos pueden ser muy útiles si no hay un escáner de puertos disponible en el sistema o si desea encadenar un escáner de puertos con los otros scripts basados en Python para su investigación.

Resumen

Este capítulo proporcionó una descripción general de los dominios de las investigaciones forenses basadas en la red y los ejemplos con Dshell y Scapy. Hemos demostrado cómo buscar conexiones HTTP sospechosas (como descargas de archivos) o cómo buscar datos filtrados a través del protocolo SMB con Dshell. En la segunda sección, creamos nuestro propio escáner de puertos con la ayuda de Scapy y lo usamos para recopilar más información sobre los sistemas potencialmente comprometidos.

Después discutimos las áreas de algoritmos forenses, los sistemas Windows y Unix, así como la capa de red, el siguiente capítulo se ocupará de los sistemas virtualizados y hipervisores que se están convirtiendo en una parte importante de cada empresa.

5

Uso de Python para la virtualización forense

Actualmente, la virtualizaci·n es uno de los conceptos m§s modernos de la IT moderna. Para el an§lisis forense, presenta nuevos desaf²os y nuevas t®cnicas. En este cap²tulo, mostraremos c·mo la virtualizaci·n presenta lo siguiente:

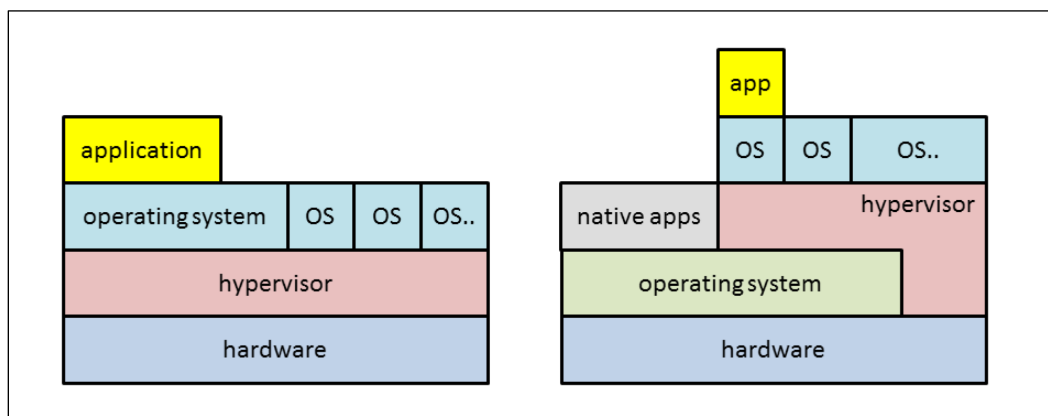
- Nuevos vectores de ataque
- Nuevas posibilidades de reunir pruebas
- Nuevos objetivos para el análisis forense, como la capa de virtualización
- Nuevas fuentes de datos forenses

Considerando la virtualización como una nueva superficie de ataque

Antes de comenzar con un análisis forense, es importante comprender qué buscar. Con la virtualización, hay nuevos vectores de ataque y escenarios que se introducen. En las siguientes secciones, describiremos algunos de los escenarios y cómo buscar la evidencia correspondiente.

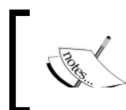
Virtualización como capa adicional de abstracción

La virtualización es la técnica de emular sistemas de IT como servidores, estaciones de trabajo, redes y almacenes. El componente responsable de la emulación del hardware virtual se define como **hypervisor**. La siguiente figura muestra los dos tipos principales de virtualización de sistemas que se usan hoy:



TLa arquitectura en el lado izquierdo se llama arquitectura **bare-metal hypervisor** y también se conoce como hipervisor de **tipo 1**. En esta arquitectura, el hipervisor reemplaza el sistema operativo y se ejecuta directamente en el hardware bare metal. Ejemplos de hipervisores de tipo I son VMware ESXi y Microsoft Hyper-V.

El lado derecho de la imagen representa una arquitectura que se suele recomendarcomo **desktop virtualization** o un hipervisor de tipo 2. En esta arquitectura, hay un sistema operativo estándar que se ejecuta en el hardware, por ejemplo, un sistema de escritorio estándar Windows 8 o Linux. El hipervisor se ejecuta entre otras aplicaciones nativas directamente en este sistema operativo. Algunas funcionalidades del hipervisor pueden interactuar directamente con el hardware subyacente, por ejemplo, proporcionando controladores especiales. Para los hipervisores de tipo 2, el sistema operativo que se ejecuta directamente en el hardware se denomina **sistema operativo host**, mientras que el sistema operativo que se ejecuta en una máquina virtual se denomina **sistema operativo invitado**. Ejemplos de arquitecturas de hipervisor Tipo 2 son Oracle VirtualBox y VMware Workstation. Estos hipervisores se pueden instalar como cualquier otra aplicación en un sistema operativo existente.



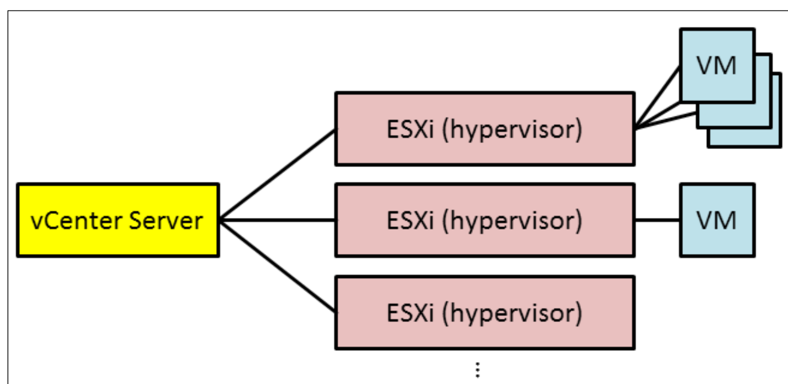
[Si bien Hyper-V parece tipo 2, en realidad convierte el sistema operativo host en otro sistema operativo huésped durante la instalación y establece una arquitectura tipo 1.]

Una característica común de casi todos los entornos de virtualización es la capacidad de crear **snapshots**. Una snapshot de un sistema virtual contiene un estado congelado en el tiempo del sistema. Todos los cambios en el sistema que ocurren después de la creación de la snapshot pueden ser deshechos por el hipervisor para retroceder hasta el momento en que se tomó la snapshot. Además, la mayoría de los sistemas permiten tener múltiples snapshot de unsistema único y retroceder y avanzar a instantáneas arbitrarias. Las snapshot se pueden utilizar como una fuente de datos forenses, que demostraremos en la sección Uso de la virtualización como fuente de evidencia.



Para los forenses, las snapshots deben ser tratadas como máquinas independientes. Si un sistema virtual está sujeto al análisis forense, siempre verifique si este sistema es un sistema virtual y si hay snapshots. Si existen snapshots, el análisis forense debe repetirse para cada snapshots como si se tratara de una máquina virtual independiente. La razón detrás de este requisito es que es muy probable que no se conozca cuando el sistema se vio comprometido, cuando el atacante intentó destruir la evidencia y lo más importante, qué versión de la máquina se estaba ejecutando durante el ataque.

La mayoría de los entornos de virtualización constan de más de un hipervisor. Para facilitar la administración de múltiples hipervisores y habilitar características adicionales; por ejemplo, movimiento de máquinas entre hipervisores para la conmutación por error, equilibrio de carga, y ahorrar energía; Estos entornos proporcionan una administración central para todos los hipervisores. En el caso de VMware vSphere, este componente de gestión se llama **vCenter Server**, como sigue:



Si se utiliza **vCenter Server**, se supone que todas las tareas administrativas deben manejarse a través de esta instancia de **vCenter Server**.

¿Cómo influye esta nueva capa de hipervisor en escenarios de ataque y análisis forense?

La introducción de la nueva capa de hipervisor también introduce una nueva capa que se puede usar para manipular sistemas virtuales sin detección y agrega otra capa nueva que puede estar sujeta a los ataques. En las siguientes secciones, proporcionaremos algunos ejemplos de escenarios para ataques que se cometen a través del hipervisor.

Creación de máquinas rogue

Si un atacante puede obtener acceso al hipervisor, puede simplemente crear nuevos recursos virtuales. Estos recursos pueden actuar como cabeza de puente en la red o simplemente robar los recursos de memoria y calcular los recursos del entorno. Por lo tanto, es crucial extraer la creación y la eliminación de recursos virtuales durante un análisis forense del entorno de hipervisor.

Afortunadamente, cada entorno de virtualización generalizado ofrece APIs y enlaces de idioma para enumerar las máquinas virtuales y otros recursos virtuales del entorno. En este capítulo, elegimos utilizar VMware vSphere como ejemplo destacado de un entorno de virtualización.



VMware vSphere es uno de los entornos de virtualización más utilizados para la virtualización en las instalaciones. Su estructura básica consiste en una instancia de administración central llamada vCenter Server y uno o varios sistemas que en realidad albergan el entorno virtual (hipervisores), llamados servidores ESXi. Para controlar mediante programación un entorno de vSphere con Python, se usa pyVmomi. Este Python SDK está disponible en Github en https://github.com/vmware/pyvmomi.

A continuación, usaremos `pyVmomi` para crear una lista de todas las máquinas virtuales. Se recomienda ejecutar dicho análisis de inventario a intervalos regulares para comparar la lista de activos virtuales existentes con su base de datos de inventario local.

Recomendamos instalar pyVmomi usando `pip`:

```
user@lab:~$ pip install --upgrade pyVmomi
```



Código de ejemplo para pyVmomi

Hay un proyecto en GitHub sobre un código de ejemplo provisto por la comunidad para `pyVmomi`. Puede obtener más información sobre estas muestras en https://vmware.github.io/pyvmomi-community-samples/.

A continuación, se puede usar un script como se muestra a continuación para enumerar todos los sistemas del entorno vSphere:

```
#!/usr/bin/env python

from pyVim import connect
from pyVmomi import vmodl
import sys

def print_vm_info(vm):
    """
    Print the information for the given virtual machine.
    If vm is a folder, recurse into that folder.
    """

    # check if this a folder...
    if hasattr(vm, 'childEntity'):
        vms = vm.childEntity
        for child in vms:
            print_vm_info(child)

    vm_info = vm.summary

    print 'Name:      ', vm_info.config.name
    print 'State:     ', vm_info.runtime.powerState
    print 'Path:      ', vm_info.config.vmPathName
    print 'Guest:     ', vm_info.config.guestFullName
    print 'UUID:      ', vm_info.config.instanceUuid
    print 'Bios UUID: ', vm_info.config.uuid
    print "----------\n"


if __name__ == '__main__':
    if len(sys.argv) < 5:
        print 'Usage: %s host user password port' % sys.argv[0]
        sys.exit(1)

    service = connect.SmartConnect(host=sys.argv[1],
                                   user=sys.argv[2],
                                   pwd=sys.argv[3],
                                   port=int(sys.argv[4]))

    # access the inventory
    content = service.RetrieveContent()
```

```
    children = content.rootFolder.childEntity

    # iterate over inventory
    for child in children:
        if hasattr(child, 'vmFolder'):
            dc = child
        else:
            # no folder containing virtual machines -> ignore
            continue

        vm_folder = dc.vmFolder
        vm_list = vm_folder.childEntity
        for vm in vm_list:
            print_vm_info(vm)
```

Este script crea una conexión a la plataforma vCenter Server. Sin embargo, también se puede usar para conectarse a una sola instancia de hipervisor ESXi. Esto es posible porque la API ofrecida al script es idéntica para ambas variantes de administración.

> La API utilizada por pyVmomi es la API del servicio web de vSphere. Una descripción detallada está disponible en el SDK de Servicios web de vSphere a través de https://www.vmware.com/support/developer/vc-sdk/.

Las líneas resaltadas muestran que la secuencia de comandos usa la recursividad para enumerar todas las máquinas virtuales. Esto es necesario porque en VMware vSphere, las máquinas virtuales pueden colocarse en grupos anidados.

Aquí hay una llamada de muestra de este script con el resultado de una sola máquina virtual:

```
user@lab:~$ python enumerateVMs.py 192.168.167.26 'readonly' 'mypwd'
443 Name: vCenterServer
State:      poweredOff
Path:      [datastore1] vCenterServer/vCenterServer.vmx
Guest:     Microsoft Windows Server 2012 (64-bit)
UUID:      522b96ec-7987-a974-98f1-ee8c4199dda4
Bios UUID: 564d8ec9-1b42-d235-a67c-d978c5107179
----------
```

El resultado muestra el nombre de la máquina virtual, su estado actual, la ruta de acceso de su archivo de configuración, una sugerencia para el sistema operativo invitado y las ID únicas para la instancia y la configuración de la BIOS. La información de la ruta es valiosa, especialmente, porque muestra dónde encontrar toda la configuración de la máquina virtual y el archivo de datos.

Clonación de sistemas

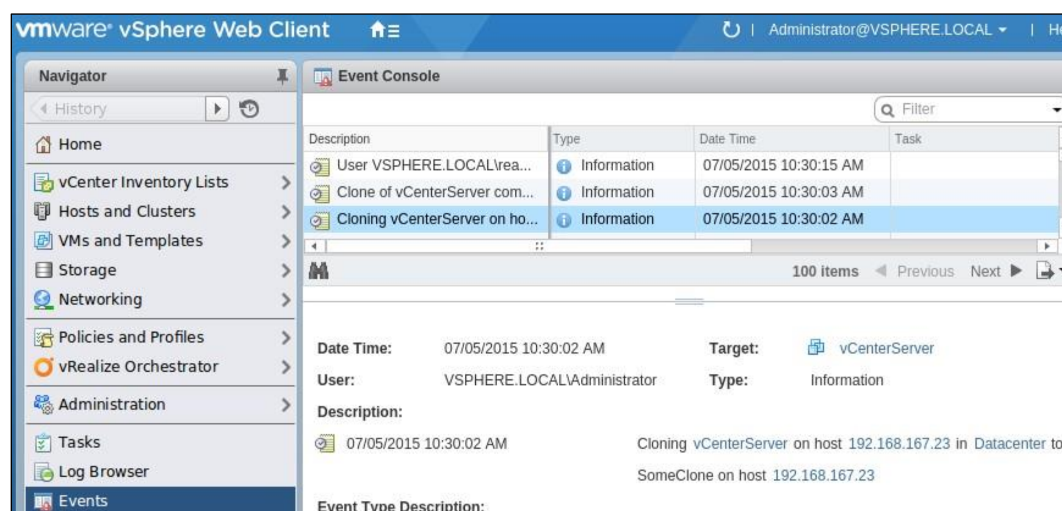
En la sección anterior, utilizamos la API del hipervisor para obtener los datos forenses. En esta sección, buscaremos rastros de abuso de esta API. Por lo tanto, analizaremos la información de registro de la instalación de vSphere.



Recopilar información de registro en un sistema de registro central

En esta sección, asumiremos que la información de registro se almacena con la configuración predeterminada de la instalación de vSphere. Sin embargo, al configurar un sistema, recomendamos almacenar la información de registro en un sistema de registro dedicado. Esto hace que sea más difícil para un atacante manipular los registros del sistema ya que requiere acceso no solo a su sistema de destino, sino también al sistema central de recopilación de registros. Otra ventaja de muchos sistemas centrales de recopilación de registros es la función integrada de análisis de registros.

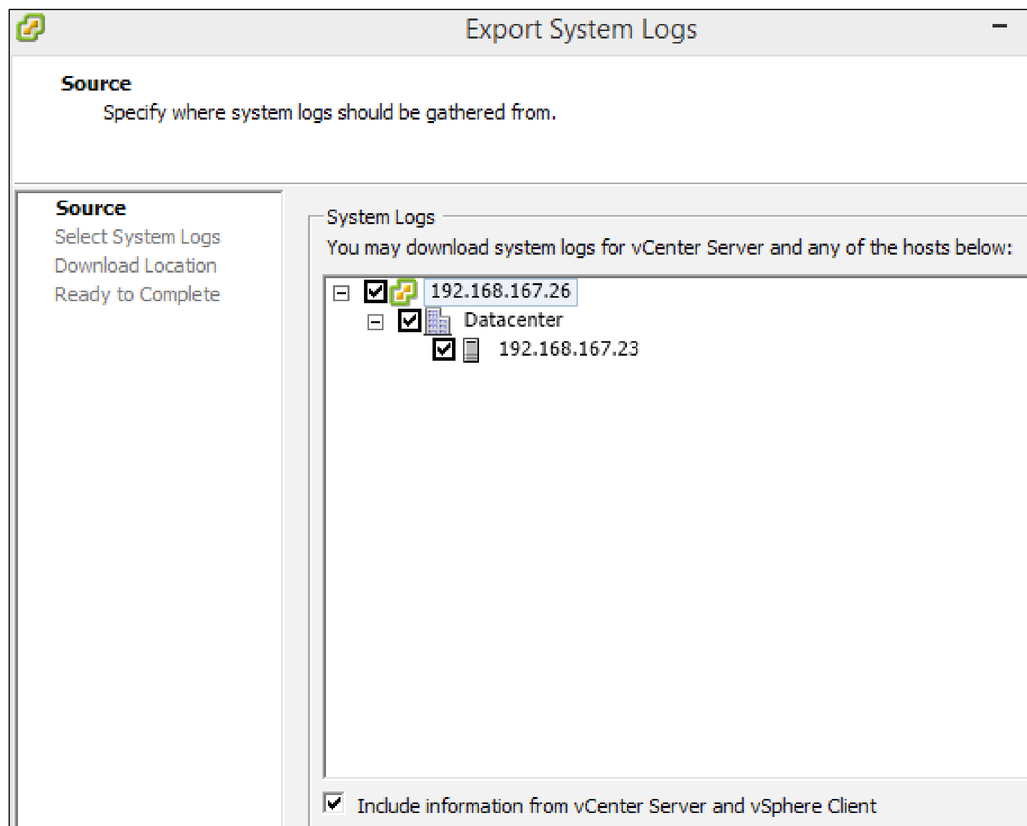
Si bien se recomienda encarecidamente una copia de todos los registros del sistema para un análisis forense solido, los eventos únicos también se pueden revisar utilizando el navegador de eventos de VMware vSphere, de la siguiente manera:



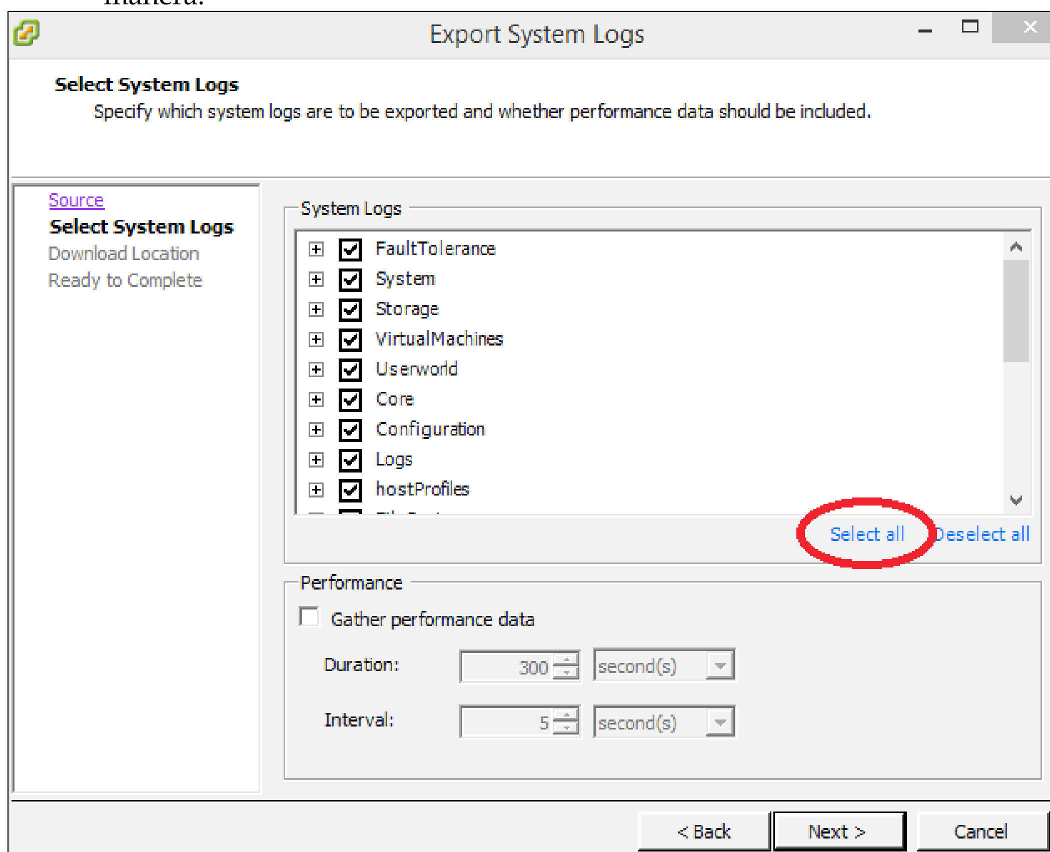
El entorno vSphere ofrece la recopilación y el almacenamiento de todos los archivos de registro en un archivo. Realice los siguientes pasos para obtener un archivo de todos los datos de registro disponibles:

- Use la versión de Windows de vSphere Web Client e inicie sesión en **vCenter Server**.
- En el menú **Administración**, seleccione **Exportar registros del sistema.**

- Seleccione uno o varios servidores de vCenter para exportar los registros, como se muestra a continuación:



- Cuando se le solicite **Seleccionar registros del sistema**, asegúrese de que todos los tipos de registro estén seleccionados de la siguiente manera:



Los archivos de registro se guardan como archivos comprimidos. Un archivo representa la información de registro de un sistema, es decir, vCenter Server o ESXi host.

Primero, extraeremos el archivo de registro recopilado usando `tar` con un comando de la siguiente manera:

```
user@lab:~$ tar xfz 192.168.167.26-vcsupport-2015-07-05@11-21-54.tgz
```

El nombre de fichero de este archivo sigue el formato Host/IP-`vcsupport` (para vCenter Server)-timestamp. El directorio en este archivo sigue el esquema de nombres vc-Hostname-Timestamp, por ejemplo, `vc-winserver-2015-07-05--02.19`. Las marcas de tiempo del nombre del archivo y el directorio contenido generalmente no coinciden. Esto puede deberse a la deriva del reloj y al tiempo requerido para transmitir y comprimir los registros.

A continuación, utilizaremos los registros de vCenter Server para reconstruir los eventos que indican la clonación de máquinas virtuales. En este ejemplo, utilizaremos la redundancia de los registros y utilizaremos los datos de registro de uno de los servicios core de vCenter Server: vpxd, es decir, el demonio de vCenter core:

```
#!/usr/bin/env python

import gzip
import os
from os.path import join
import re
import sys


# used to map session IDs to users and source IPs
session2user_ip = {}

def _logopen(filename):
    """Helper to provide transparent decompressing of compressed
       logs, if indicated by the file name.
    """
    if re.match(r'.*\.gz', filename):
        return gzip.open(filename, 'r')

    return open(filename, 'r')

def collect_session_data(vpxlogdir):
    """Uses vpx performance logs to map the session ID to
       source user name and IP"""
    extract = re.compile(r'SessionStats/SessionPool/Session/
Id=\'([^\']+)\'/Username=\'([^\']+)\'/ClientIP=\'([^\']+)\'')

    logfiles = os.listdir(vpxlogdir)
    logfiles = filter(lambda x: 'vpxd-profiler-' in x,
    logfiles) for fname in logfiles:
        fpath = join(vpxlogdir, fname)
        f = _logopen(fpath)

        for line in f:
            m = extract.search(line)
            if m:
```

```
                session2user_ip[m.group(1)] = (m.group(2), m.group(3))

        f.close()

def print_cloning_hints(basedir):
    """Print timestamp, user, and IP address for VM cloning without
       by reconstructing from vpxd logs instead of accessing
       the 'official' event logs"""
    vpxlogdir = join(basedir, 'ProgramData',
                              'vCenterServer',
                              'logs',
                              'vmware-vpx')
    collect_session_data(vpxlogdir)

    extract = re.compile(r'^([^ ]+).*BEGIN task-.*?vim\.
VirtualMachine\.clone -- ([0-9a-f-]+).*')

    logfiles = os.listdir(vpxlogdir)
    logfiles = filter(lambda x: re.match('vpxd-[0-9]+.log(.gz)?', x),
logfiles)
    logfiles.sort()

    for fname in logfiles:
        fpath = join(vpxlogdir, fname)
        f = _logopen(fpath)

        for line in f:
            m = extract.match(line)
            if m == None:
                continue

            timestamp = m.group(1)
            session = m.group(2)
            (user, ip) = session2user_ip.get(session,
('***UNKNOWN***', '***UNKNOWN***'))
            print 'Hint for cloning at %s by %s from %s' % (timestamp,
user, ip)

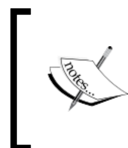
if __name__ == '__main__':
    if len(sys.argv) < 2:
        print 'Usage: %s vCenterLogDirectory' % sys.argv[0]
        sys.exit(1)

    print_cloning_hints(sys.argv[1])
```

Primero, este script lee el llamado registro de rendimiento de vpxd. Este registro contiene datos sobre sesiones de clientes y lo utilizamos para extraer una asignación del identificador de sesión único al nombre de usuario del cliente y la dirección IP desde la que se conecta el cliente. En el segundo paso, se busca el registro principal de `vpxd` para el inicio de tareas de vim. Tipo `VirtualMachine.clone`, es decir, la clonación de máquinas virtuales en el lado del servidor. La información de la sesión se busca en la asignación que se extrae del registro de rendimiento para recuperar los datos sobre posibles eventos de clonación, de la siguiente manera:

```
user@lab:~$ python extractCloning.py vc-winserver-2015-07-05--
02.19/ Hint for cloning at 2015-07-05T01:30:01.071-07:00 by
VSPHERE.LOCAL\
Administrator from 192.168.167.26
```

En el ejemplo, el script reveló que la cuenta de `administrador` se utilizó para clonar una máquina virtual. Esta sugerencia se puede correlacionar con el registro de eventos de vCenter Server y aparecerá allí también. Si no es así, este es un fuerte indicador de un entorno comprometido.



Dependiendo del entorno de su sistema, las operaciones como clonar y exportar máquinas virtuales pueden ser parte de las operaciones diarias. En ese caso, la secuencia de comandos anterior o sus variantes pueden usarse para detectar usuarios inusuales o IP de origen que realizan estas operaciones.

Se pueden usar búsquedas y correlaciones similares para otros eventos de interés. La copia de los archivos del almacén de datos o la exportación de máquinas virtuales son candidatos prometedores.

Búsqueda de mal uso de recursos virtuales

No es solo el atacante motivado lo que estamos buscando. Con la virtualización, también está el administrador legítimo de la infraestructura virtual que hace su vida más fácil doblando algunas reglas. Además, un atacante puede usar el poder de la virtualización para remodelar la topología de la infraestructura según sus necesidades. En las siguientes secciones, mostraremos algunos escenarios y métodos de detección.

Detección de interfaces de red deshonestas

La virtualización de red permite que las operaciones creen infraestructuras de red casi arbitrarias en una red física estática. Esta capacidad a veces se denomina **centro de datos como servicio (DCaaS)**. DCaaS permite a los clientes utilizar una porción definida de un centro de datos físico para definir centros de datos virtuales en el software.

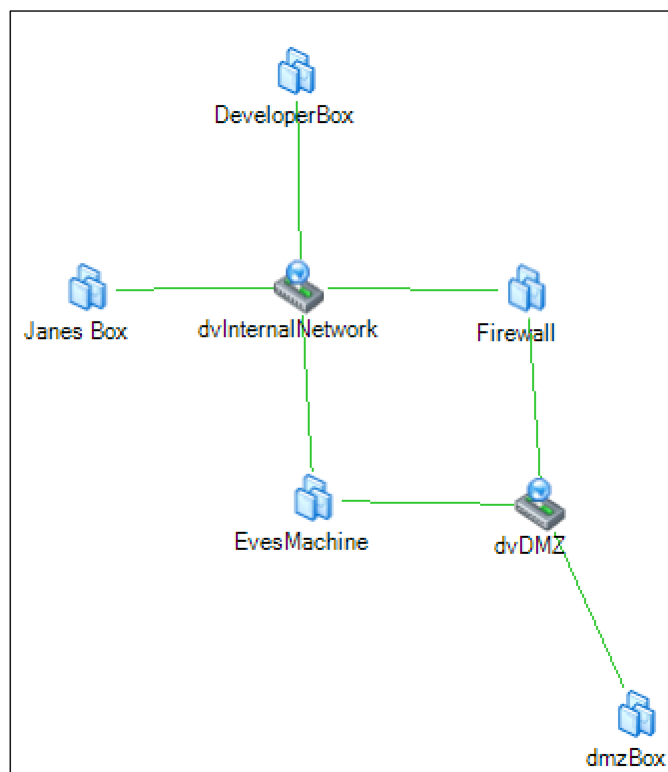
Debido al acceso malicioso a esta capacidad o a un error humano, la configuración de red resultante puede exponer los recursos internos a Internet, evitar los cortafuegos o permitir el acceso a servicios maliciosos.

Por lo tanto, vamos a mostrar una forma sencilla de obtener mediante programación la configuración de red de un entorno vSphere usando Python.



Visualice redes virtuales

La mayoría de los entornos de virtualización tienen capacidades incorporadas para visualizar la configuración de la red virtual. Por ejemplo, VMware vSphere puede crear una imagen de la topología de red. En un análisis forense, esto puede servir como punto de partida y soporte, centrando el siguiente paso en los activos más prometedores.



Esta imagen se generó con el software de cliente de Windows para VMware vCenter Server y muestra nuestra configuración de prueba. Obviamente, **EvesMachine** no está conectado correctamente, es decir, puede pasar por alto el **Firewall.**

Los scripts de ejemplo de la comunidad para `pyVmomi` ya proporcionan un script para iterar sobre todas las interfaces de red, `https://github.com/vmware/pyvmomi-community-samples/blob/master/samples/getvnicinfo.py`, y muestran las conexiones de las máquinas virtuales. Por lo tanto, modificamos este script para mostrar solo las máquinas virtuales que tienen conexiones de red múltiples, de la siguiente manera:

```
#!/usr/bin/env python

from pyVim import connect
from pyVmomi import vmodl
from pyVmomi import vim
import sys

def generate_portgroup_info(content):
    """Enumerates all hypervisors to get
       network infrastructure information"""
    host_view =
content.viewManager.CreateContainerView(content. rootFolder,
                                        [vim.HostSystem],
                                        True)
    hostlist = [host for host in host_view.view]
    host_view.Destroy()

    hostPgDict = {}
    for host in hostlist:
        pgs = host.config.network.portgroup
        hostPgDict[host] = pgs

    return (hostlist, hostPgDict)

def get_vms(content, min_nics=1):
    vm_view =
content.viewManager.CreateContainerView(content. rootFolder,
                                        [vim.VirtualMachine],
                                        True)
    vms = [vm for vm in vm_view.view]
    vm_view.Destroy()

    vm_with_nics = []
    for vm in vms:
        num_nics = 0
        for dev in vm.config.hardware.device:
            # ignore non-network devices
            if not isinstance(dev, vim.vm.device.VirtualEthernetCard):
```

```
                continue

            num_nics = num_nics + 1
            if num_nics >= min_nics:
                vm_with_nics.append(vm)
                break

    return vm_with_nics

def print_vm_info(vm, hosts, host2portgroup, content):
    print "\n=== %s ===" % vm.name

    for dev in vm.config.hardware.device:
        if not isinstance(dev, vim.vm.device.VirtualEthernetCard):
            continue

        dev_backing = dev.backing
        if hasattr(dev_backing, 'port'):
            # NIC is connected to distributed vSwitch
            portGroupKey = dev.backing.port.portgroupKey
            dvsUuid = dev.backing.port.switchUuid
            try:
                dvs = content.dvSwitchManager.QueryDvsByUuid(dvsUuid)
            except:
                portGroup = 'ERROR: DVS not found!'
                vlanId = 'N/A'
                vSwitch = 'N/A'
            else:
                pgObj = dvs.LookupDvPortGroup(portGroupKey)
                portGroup = pgObj.config.name
                vlObj = pgObj.config.defaultPortConfig.vlan
                if hasattr(vlObj, 'pvlanId'):
                    vlanId = str(pgObj.config.defaultPortConfig.vlan.
pvlanId)
                else:
                    vlanId = str(pgObj.config.defaultPortConfig.vlan.
vlanId)
                vSwitch = str(dvs.name)
        else:
            # NIC is connected to simple vSwitch
            portGroup = dev.backing.network.name
            vmHost = vm.runtime.host

            # look up the port group from the
```

```
            # matching host
            host_pos = hosts.index(vmHost)
            viewHost = hosts[host_pos]
            pgs = host2portgroup[viewHost]

            for p in pgs:
                if portgroup in p.key:
                    vlanId = str(p.spec.vlanId)
                    vSwitch = str(p.spec.vswitchName)

        if portGroup is None:
            portGroup = 'N/A'

        print '%s -> %s @ %s -> %s (VLAN %s)' % (dev.deviceInfo.label,
                                                 dev.macAddress,
                                                 vSwitch,
                                                 portGroup,
                                                 vlanId)

def print_dual_homed_vms(service):
    """Lists all virtual machines with multiple
       NICs to different networks"""

    content = service.RetrieveContent()
    hosts, host2portgroup = generate_portgroup_info(content)
    vms = get_vms(content, min_nics=2)
    for vm in vms:
        print_vm_info(vm, hosts, host2portgroup, content)


if   name__ == '  main  ':
    if len(sys.argv) < 5:
        print 'Usage: %s host user password port' % sys.argv[0]
        sys.exit(1)

    service = connect.SmartConnect(host=sys.argv[1],
                                   user=sys.argv[2],
                                   pwd=sys.argv[3],
                                   port=int(sys.argv[4]))
    print_dual_homed_vms(service)
```

En primer lugar, este script itera sobre todos los hosts (hipervisor) para recopilar información sobre los conmutadores virtuales que están presentes en cada sistema ESXi. Luego, itera sobre todas las máquinas virtuales para recopilar aquellas con más de una tarjeta de red. Luego, la información sobre las tarjetas de red virtuales se combina con la información sobre los conmutadores virtuales para derivar la información sobre la conectividad.

Aquí está el resultado de muestra de nuestro entorno de laboratorio como se describió anteriormente:

```
user@lab:~$ python listDualHomed.py 192.168.167.26 readonly 'mypwd' 443
=== EvesMachine ===
Network adapter 1 -> 00:50:56:ab:04:38 @ dvSwitch ->
dvInternalNetwork (VLAN 8)
Network adapter 2 -> 00:50:56:ab:23:50 @ dvSwitch -> dvDMZ (VLAN 0)

=== Firewall ===
Network adapter 1 -> 00:50:56:ab:12:e6 @ dvSwitch ->
dvInternalNetwork (VLAN 8)
Network adapter 2 -> 00:50:56:ab:4b:62 @ dvSwitch -> dvDMZ (VLAN 0)
```

Nuestro script identificó correctamente los dos sistemas, `EvesMachine` y `Firewall`, conectados simultáneamente a diferentes redes. En este caso particular, ambos sistemas se pueden usar para conectar `VLAN 0` con `VLAN 8` en el mismo conmutador virtual, `dvSwitch`.

Detección de acceso directo al hardware

Puede parecer un oxímoron, pero la mayoría de las técnicas de virtualización permiten el acceso directo al hardware. Las razones legítimas para permitir que los sistemas virtuales accedan directamente a una pieza de hardware sin tener que usar los servicios del hipervisor son las siguientes:

- **Se supone que el hardware especial debe estar conectado a una máquina virtual:** hardware especial como relojes de radio para servidores de tiempo virtual o dongles que forman parte de un mecanismo de protección contra copia.
- **Uso temporal de medios físicos en un sistema virtual:** a veces, esta capacidad se usa para acceder a medios desde sistemas físicos desde un entorno virtual, por ejemplo, para restaurar copias de seguridad de un medio físico a un sistema virtual. En general, los sistemas de almacenamiento conectados a la red deberían preferirse a la conexión de medios físicos a un sistema virtual.
- **Uso permanente de las unidades de un hipervisor desde una máquina virtual:** esto puede ser útil si el sistema virtual usa software que se proporciona en medios físicos y, por lo tanto, necesita acceso a una unidad física real para la instalación y actualizaciones del software. Sin embargo, uno debería considerar usar versiones descargadas o imágenes ISO en lugar de otorgar acceso directo al hardware del hipervisor.

Como puede adivinar, de acuerdo con esta lista, el acceso directo al hardware es más la excepción que la regla en un centro de datos virtualizado moderno. Además, el acceso directo al hardware del hipervisor rompe un principio fundamental de la virtualización.

> El acceso directo al hardware evita el mecanismo de seguridad del entorno de virtualización, es decir, todo el hardware virtual está controlado por el hipervisor. En consecuencia, el acceso directo al hardware siempre plantea el riesgo de manipulación de recursos del hipervisor, pérdida de datos e inestabilidades del sistema.

Los siguientes son algunos ejemplos del hardware directamente conectado que probablemente sea malicioso:

- Dispositivos de red (crear conexiones de red que son invisibles para el hipervisor)
- Teclado, mouse, etc. (crear accesos a la consola que son invisibles para el hipervisor)
- Particiones de disco de hipervisor

Esto último es especialmente peligroso. Si una máquina virtual logra obtener acceso al disco raw de hipervisor, puede manipular el entorno de virtualización. Las consecuencias incluyen el control total sobre el entorno de virtualización junto con el acceso a todas las máquinas virtuales, todas las redes virtuales, la capacidad de crear nuevos recursos deshonestos y remodelar la topología general de la red.

> Para VMware vSphere, el acceso directo al hardware se almacena en la configuración de las máquinas virtuales. En consecuencia, la importación de una máquina virtual a partir de un origen desconocido o no confiable (en el formato nativo de vSphere) puede crear acceso de hardware no fiable.

El siguiente script se conecta a una instancia de VMware vSphere y enumera todas las máquinas virtuales con acceso directo al hardware:

```
#!/usr/bin/env python

from pyVim import connect
from pyVmomi import vmodl
from pyVmomi import vim
import re
import sys


def get_vms(content):
```

```
    """Returns a list of all virtual machines."""
    vm_view = content.viewManager.CreateContainerView(content.
rootFolder,
                                                      [vim.
VirtualMachine],
                                                      True)
    vms = [vm for vm in vm_view.view]
    vm_view.Destroy()
    return vms

def print_vm_hardware_access(vm):
    findings = []

    for dev in vm.config.hardware.device:
        if isinstance(dev, vim.vm.device.VirtualUSB):
            findings.append('USB access to host device ' + dev.
backing.deviceName)
        elif isinstance(dev, vim.vm.device.VirtualSerialPort):
            findings.append('Serial port access')
        elif isinstance(dev, vim.vm.device.VirtualCdrom):
            if not dev.backing is None:
                if 'vmfs/devices/cdrom' in dev.backing.deviceName:
                    findings.append('Access to CD/DVD drive')
        elif isinstance(dev, vim.vm.device.VirtualDisk):
            if dev.backing is None or \
               dev.backing.fileName is None or \
               re.match(r'.*\.vmdk', dev.backing.fileName) is None:
               findings.append('Suspicious HDD configuration')

    if len(findings) > 0:
        print '=== %s hardware configuration findings ===' % vm.name
        for l in findings:
            print l
        print "\n"

def print_direct_hardware_access(content):
    vms = get_vms(content)
    for vm in vms:
        print_vm_hardware_access(vm)


if __name__ == '__main__':
    if len(sys.argv) < 5:
        print 'Usage: %s host user password port' % sys.argv[0]
```

```
        sys.exit(1)

    service = connect.SmartConnect(host=sys.argv[1],
                                   user=sys.argv[2],
                                   pwd=sys.argv[3],
                                   port=int(sys.argv[4]))

    # access the inventory
    content = service.RetrieveContent()
    print_direct_hardware_access(content)
```

Este script está muy ansioso, es decir, no comprueba si el dispositivo está realmente en un estado conectado o si hay medios accesibles a través del dispositivo. Sin embargo, un resultado similar al siguiente requiere una inspección más profunda:

```
user@lab:~$ python listHardwareAccess.py 192.168.167.26 readonly pwd 443
=== EvesMachine hardware configuration findings ===
Access to CD/DVD drive
Serial port access
USB access to host device path:2/0 version:2

=== DeveloperBox hardware configuration findings ===
Access to CD/DVD drive

=== dmzBox hardware configuration findings ===
Access to CD/DVD drive
```

EvesMachine parece tener acceso directo a un dispositivo USB conectado a su sistema de hipervisor. Además, parece haber un enlace directo al puerto serie del hipervisor. El acceso a la unidad de CD/DVD del hipervisor no debe otorgarse en general. Sin embargo, para muchas instalaciones, las personas tienden a usar la unidad óptica del hipervisor para instalar o actualizar un software.



Extraiga la configuración de hardware del archivo VMX

El uso de un script como el anterior requiere acceso al entorno virtual. Por lo tanto, el propósito principal de tales scripts es reducir el foco de la investigación forense. Para evidencia y registro permanente, el directorio de las máquinas virtuales debe copiarse del almacén de datos. Allí, el archivo VMX contiene todas las configuraciones de VM específicas, incluido el acceso al hardware.

En esta y en las secciones anteriores, la virtualización se considera una superficie de ataque adicional. En la siguiente sección, describiremos cómo las técnicas de virtualización pueden realmente respaldar una investigación forense.

Usar la virtualización como fuente de evidencia

La virtualización no solo es peligrosa y desafiante en lo que respecta a las investigaciones forenses, también existe el potencial de utilizar la virtualización como una herramienta para reunir evidencia forense. En las siguientes secciones, verá varias fuentes que pueden conducir a la evidencia.

Creación de copias forenses de contenido de RAM

Normalmente, la creación de una copia del contenido de RAM de un sistema requiere acceso al sistema de destino, un inicio de sesión, la instalación de las herramientas necesarias y la copia de la descarga de RAM en un medio externo. Todos estos pasos son intrusivos, es decir, cambian el estado del sistema y están sujetos a la detección por parte del atacante o su malware. Además, un atacante con privilegios administrativos puede ocultar partes de la memoria del sistema de los volcados de memoria, por ejemplo, manipulando la asignación de memoria y los algoritmos de protección.

Para superar las desventajas de este método, la capa del hipervisor se puede utilizar para obtener una copia completa y no alterada de la memoria de un sistema virtual. El siguiente script se puede usar para crear una snapshot que incluya el contenido de RAM de una máquina virtual:

```
#!/usr/bin/env python

from pyVim import connect
from pyVmomi import vim
from datetime import datetime
import sys

def make_snapshot(service, vmname):
    """Creates a snapshot of all virtual machines with the given
name"""

    snap_name = 'Memory_Snapshot'
```

```
    snap_desc = 'Snapshot for investigation taken at ' + datetime.
now().isoformat()

    content = service.RetrieveContent()
    vm_view = content.viewManager.CreateContainerView(content.
rootFolder,
                                                      [vim.
VirtualMachine],
                                                      True)
    vms = [vm for vm in vm_view.view if vm.name==vmname]
    vm_view.Destroy()

    for vm in vms:
        print 'Taking snapshot from VM UUID=%s' % vm.summary.config.
uuid
        vm.CreateSnapshot_Task(name = snap_name,
                               description = snap_desc,
                               memory = True,
                               quiesce=False)
        print "Done.\n"


if __name__ == '__main__':
    if len(sys.argv) < 6:
        print 'Usage: %s host user password port vmname' % sys.argv[0]
        sys.exit(1)

    service = connect.SmartConnect(host=sys.argv[1],
                                   user=sys.argv[2],
                                   pwd=sys.argv[3],
                                   port=int(sys.argv[4]))


    make_snapshot(service, sys.argv[5])
```

Este script busca máquinas virtuales con el nombre especificado y crea una snapshot. El parámetro resaltado hace que vSphere escriba los contenidos de RAM de la máquina virtual en el almacén de datos junto con los demás archivos de datos de snapshot.

Estos volcados de RAM residen en el directorio de la máquina virtual. El script de enumeración en este capítulo muestra la ruta a este directorio. Además, vSphere Client permite explorar y descargar el almacén de datos de la máquina virtual.

Los contenidos de RAM se almacenan en un archivo con la extensión `.vmem`, por ejemplo, `EvesMachine-Snapshot2.vmem`.

Uso de snapshots como imágenes de disco

Para los sistemas físicos, la creación de una imagen de disco forense generalmente incorpora desconectar el sistema, apagarlo, extraer el disco duro y copiarlo. Obviamente, el sistema no está operativo durante este procedimiento y, como consecuencia, los propietarios de negocios son muy reacios a conceder estos tiempos de inactividad debido a una vaga sospecha de un posible compromiso.

Por otro lado, la creación de una snapshot de una máquina virtual no genera básicamente tiempo de inactividad, pero el resultado es una imagen de disco forense del activo virtual.



¡Siempre verifica si un sistema es virtual!

Como la creación de datos forenses es mucho más fácil para los sistemas virtuales que para los sistemas físicos, uno de los primeros pasos en una investigación forense debería ser verificar si el sistema objetivo es virtual.

La creación de la snapshot es igual al script en la sección anterior. Para VMware vSphere 5, todos los archivos deben copiarse desde el directorio del almacén de datos del hipervisor para obtener un volcado completo de los discos duros. Si el sistema virtual aún se está ejecutando, es posible que algunos archivos no se copien ya que el hipervisor no permitirá el acceso de lectura mientras estos archivos estén en uso. Normalmente, esto no es un problema ya que la snapshot solo necesita estos archivos, es decir, todos los cambios desde la creación de la snapshot se almacenan en archivos especiales de snapshot.

En VMware vSphere 6, el mecanismo de snapshot ha sido cambiado. En lugar de escribir cambios de disco en los archivos de snapshot, los cambios realizados después de la creación de la snapshot se escriben directamente en los archivos que representan los discos duros virtuales. Los archivos de snapshot se utilizan para preservar el contenido original de las unidades de disco (comportamiento de copiado sobre escritura).

Por lo tanto, los archivos que se copiarán de un entorno de VMware vSphere 6 contendrán todas las entradas del directorio de la máquina virtual.

Para el análisis forense, las imágenes de disco capturadas se pueden conectar a una estación de trabajo forense virtual. Allí, estas imágenes se pueden tratar como cualquier otro disco duro físico. Por supuesto, las copias originales deben permanecer intactas para proporcionar solidez forense.

Captura de tráfico de red

El entorno de virtualización no solo representa las máquinas virtuales y la **tarjeta de interfaz de red (NIC)**, sino también los dispositivos de red virtual que se necesitan para interconectar estos sistemas. Esta combinación se puede utilizar para recopilar todo el tráfico de red de una red virtual al agregar un puerto de monitoreo al conmutador virtual y conectar un sistema a él, que puede capturar todo el tráfico de la red.



Si un sistema virtual en VMware vSphere puede cambiar una NIC a un modo promiscuo, esto convertirá automáticamente el puerto del conmutador correspondiente al modo de monitoreo.

Además, las ediciones empresariales de VMware vSphere proporcionan una versión avanzada de un conmutador virtual denominado **vSphere Distributed Switch (VDS)**. Este conmutador puede actuar más como un conmutador físico y proporcionar la duplicación de los puertos seleccionados a un puerto definido para el análisis del tráfico. Además, este conmutador también puede proporcionar registros de NetFlow a un puerto definido.

Para el conmutador virtual estándar, se requieren los siguientes pasos para monitorear el tráfico de la red:

- Cree un nuevo grupo de puertos en este interruptor para monitorear. Si bien esto no es estrictamente obligatorio, es muy recomendable. Sin un grupo de puertos dedicado para monitorear, todos los sistemas virtuales en el conmutador podrían monitorear todo el tráfico del conmutador.
- Modifique las configuraciones de **Seguridad** de este grupo de puertos y cambie el **modo Promiscuo** a **Aceptar**.
- Configure la tarjeta de red del sistema de captura virtual en el nuevo grupo de puertos. Este sistema ahora puede capturar todo el tráfico de red de este interruptor.

Los pasos exactos pueden diferir entre los tipos de conmutadores virtuales y sus versiones. Sin embargo, el mensaje central es que los entornos de virtualización pueden facilitar esta tarea de captura de tráfico de red. Además, los conmutadores físicos y virtuales tienen comportamientos diferentes, por ejemplo, pueden reaccionar a los cambios de configuración de las tarjetas de red conectadas.

En el siguiente capítulo, veremos cómo generar y analizar este tráfico de red capturado.

Resumen

En este capítulo, describimos cómo la virtualización cambia el panorama no solo para las operaciones de IT, sino también para el atacante y el especialista forense. Los sistemas se pueden crear, reformar y copiar por buenas y malas razones.

Proporcionamos ejemplos de cómo detectar comportamientos o configuraciones posiblemente maliciosos en el entorno de virtualización de vSphere. Además, demostramos cómo la virtualización puede ser beneficiosa para obtener volcados de RAM sin modificar de los sistemas que deberían analizarse. En el siguiente capítulo, verá ejemplos sobre cómo analizar estos volcados de RAM.

Con este conocimiento, ahora está preparado para analizar y utilizar entornos virtuales en sus análisis forenses.

6

Uso de Python para forense móvil

Mientras que el análisis forense del hardware estándar de la computadora, como los discos duros, se ha convertido en una disciplina estable con mucho trabajo de referencia, como el libro `File System Forensic Analysis`, por `Brian Carrier`, `Addison-Wesley Professional` y nuestros capítulos anteriores, todavía hay mucho debate sobre las técnicas para analizar hardware no estándar o evidencia transitoria. A pesar de su creciente papel en las investigaciones digitales, los teléfonos inteligentes aún no se consideran estándar debido a su heterogeneidad. En todas las investigaciones, es necesario seguir los principios forenses básicos. Los dos principios principales de las investigaciones forenses son los siguientes:

- Se debe tener mucho cuidado para que la evidencia sea manipulada o modificada lo menos posible.
- El curso de una investigación digital debe ser comprensible y estar abierto al escrutinio. En el mejor de los casos, los resultados de la investigación deben ser reproducibles por investigadores independientes.

El primer principio, especialmente, es un desafío en el caso de los teléfonos inteligentes, ya que la mayoría de ellos emplea sistemas operativos específicos y métodos de protección de hardware que impiden el acceso irrestricto a los datos en el sistema.

La preservación de datos de discos duros es, en la mayoría de los casos, un procedimiento simple y bien conocido. Un investigador saca el disco duro de la computadora o computadora portátil, lo conecta a su estación de trabajo con la ayuda de un bloqueador de escritura (por ejemplo, Tableau TK35) y comienza a analizarlo con soluciones de software conocidas y certificadas. Al comparar esto con el mundo de los teléfonos inteligentes, queda claro que no existe tal procedimiento. Casi todos los teléfonos inteligentes tienen su propia manera de construir su almacenamiento y continuar con esto, para cada teléfono inteligente, el investigador necesita su propia manera de obtener el volcado del almacenamiento. Si bien es muy difícil obtener los datos de un teléfono inteligente, se pueden obtener muchos más datos con referencia a la diversidad de los datos. Los teléfonos inteligentes almacenan, además de los datos habituales (por ejemplo, imágenes y documentos), datos tales como las coordenadas GPS y la posición de una celda móvil a la que el teléfono inteligente estaba conectado antes de que se apagara.

Teniendo en cuenta las oportunidades resultantes, resulta que vale la pena el gasto extra para un investigador.

En este capítulo, cubriremos los siguientes temas:

- El modelo de investigación de `Eoghan Casey` adoptado por los teléfonos inteligentes
- El análisis de los teléfonos inteligentes Android (tanto manuales como automáticos a través de **Android Data Extractor Lite (ADEL)**)
- El análisis de los teléfonos inteligentes con iOS

El modelo de investigación para teléfonos inteligentes

El **Modelo de Proceso de Investigación** de `Eoghan Casey`, también conocido como el **Modelo de Escalera**, proporciona una guía práctica y metódica paso a paso para realizar una investigación digital efectiva. Este modelo se representa como una secuencia de escaleras ascendentes que comienza en la alerta de incidente o acusación y terminan en el testimonio. Los pasos deben ser lo más genéricos posible. Este modelo intenta fusionar los deberes policiales y las tareas de los expertos forenses. Los siguientes puntos explican cada paso del Modelo de Proceso de Investigación y la diferencia entre tratar con teléfonos inteligentes y computadoras:

- **Alertas de incidentes o acusaciones:** la acusación es la señal de inicio para todo el proceso. En esta fase, se evalúan las fuentes y se solicitan consultas detalladas.

- **Evaluación del valor:** en el ámbito de la evaluación del valor, el interés del enjuiciamiento se compara con los costos en que se incurriría para enjuiciar la acción penal. Para las empresas, esto a menudo resulta en una decisión contra el enjuiciamiento (para incidentes menores, al menos). Las ventajas de un procesamiento radican en la posible compensación, la mejora de la propia seguridad y ciertos efectos de disuasión. Las desventajas de un enjuiciamiento son la necesidad de recursos, el posible tiempo de inactividad durante el cual los sistemas investigados no se pueden usar productivamente, y la mayoría de las veces un efecto de dispersión pública negativa.
- **Incidentes o protocolos de la escena del crimen**: en la criminalística clásica, a menudo se exige que la escena del crimen esté espaciosamente cerrada. **Eoghan Casey** expresa esto como lo siguiente:

  > "*Congelar*" *la evidencia en su lugar y proporcionar* "*verdad fundamental para todas las actividades que siguen*".

  Para diferentes tipos de rastros digitales, se debe verificar de forma individual cómo se define exactamente el proceso de congelación. En general, es cierto que el riesgo de cambiar las huellas debe minimizarse. Para los teléfonos inteligentes, esto significa que deben colocarse en una bolsa de Faraday que está conectada a una fuente de alimentación externa.

- **Identificación o incautación:** durante una confiscación tradicional, todos los objetos y sujetos que podrían actuar como evidencia son recogidos. Aquí, es importante que no se realicen cambios en la evidencia. Además, el ambiente de evidencia podría ser de gran relevancia. Simultáneamente a la incautación, comienza la cadena de custodia. Un documento recomendado sobre la incautación es el folleto, `Investigación electrónica de la escena del crimen:una guía para los primeros respondedores`, publicado por el Departamento de Justicia de los Estados Unidos. Este folleto proporciona consejos precisos y detallados para el personal no técnico. Otra buena fuente es el documento, `Búsqueda y decomiso de computadoras y Obtención de pruebas electrónicas en investigaciones penales`, también publicado por el Departamento de Justicia de los Estados Unidos.

- **Preservación:** Al asegurar la evidencia, se debe garantizar que no se modifiquen. Esta es la razón por la cual toda la evidencia está documentada, fotografiada, sellada y luego encerrada. En el caso de la evidencia digital, esto significa que primero se crean copias de la evidencia; la investigación adicional se hace solo en las copias. Para probar la autenticidad de las copias de la evidencia, se utilizan funciones hash criptográficas. En la mayoría de los casos, esta es la parte más difícil en el análisis forense de teléfonos móviles debido a que la creación de copias uno a uno no es posible para algunos tipos de teléfonos. Mostraremos, en la siguiente sección, cómo crear copias de seguridad que se pueden usar durante la investigación.

- **Recuperación:** `Eoghan Casey` describe la recuperación como si `arrojara una red grande`. En particular, esta fase incluye la recuperación de evidencia que ha sido eliminada, escondida, enmascarada o hecha inaccesible de cualquier otra manera. Se recomienda que haga uso de sinergias con otras pruebas. Por ejemplo, es razonable comprobar si se ha encontrado una nota con contraseñas en la escena del crimen en caso de que sea necesario leer los datos encriptados.
- **Cosecha:** Durante el análisis de la evidencia, una organización bien estructurada con una gran cantidad de datos que se necesitan. Por este motivo, primero se deben investigar los metadatos en lugar de los datos reales. Por ejemplo, los datos se pueden agrupar según el tipo de archivo o el tiempo de acceso. Esto lleva directamente a la siguiente fase, la reducción.
- **Reducción:** la tarea de reducción consiste en eliminar datos irrelevantes. Uno puede usar los metadatos por esta razón, también. Por ejemplo, los datos se pueden reducir de acuerdo con el tipo de datos. Un escenario adecuado sería reducir todos los datos a datos de imagen, solo si la acusación permite este procedimiento. El resultado de esta fase es, según `Eoghan Casey`:

  > *El conjunto más pequeño de información digital que tiene el mayor potencial para contener datos de valor probatorio.*

  Esto significa encontrar la menor cantidad de datos que tenga la mayor probabilidad de ser relevante y probatorio. En este contexto, las bases de datos hash de archivos conocidos, como **The National Software Reference Library (NIST)**, son útiles para excluir archivos ya conocidos (describimos el uso de esta biblioteca en el Capítulo 2, Algoritmos forenses).

- **Organización y búsqueda**: los aspectos de la organización están estructurando y habilitando datos para escanear. Por lo tanto, a menudo se crean índices y resúmenes o se ordenan los archivos en directorios significativos. Esto simplifica la referencia de los datos en los siguientes pasos.
- **Análisis:** esta fase incluye el análisis detallado sobre el contenido del archivo. Entre otros, se deben establecer conexiones entre los datos y las personas para determinar la persona responsable. Además, la evaluación del contenido y el contexto se realiza de acuerdo con los medios, la motivación y la oportunidad. En este paso, los experimentos son útiles para determinar el comportamiento no documentado y desarrollar nuevos métodos. Todos los resultados deben ser probados y deben ser verificables con la metodología científica.
- **Informes:** el informe no solo presenta los resultados, sino que también demuestra cómo se ha llegado a los resultados establecidos. Para esto, todas las normas y estándares considerados deben estar documentados. Además, todas las conclusiones extraídas deben estar justificadas y los modelos de explicación alternativa deben discutirse.

- **Persuasión y testimonio:** Finalmente, se trata del testimonio de una autoridad sobre el tema en la corte. El aspecto más importante es la confiabilidad de la autoridad. Una audiencia contraria a la tecnología o analogías difíciles, por ejemplo del abogado defensor, pueden ser problemáticas.

Al observar el proceso descrito anteriormente, uno puede ver solo pequeños cambios cuando se trata de teléfonos inteligentes a diferencia de otros tipos de pruebas. Sin embargo, es muy importante que un investigador comprenda en qué pasos debe tener especial cuidado.

Android

El primer sistema operativo móvil que examinaremos con la ayuda de Python es Android. En la primera subsección, demostraremos cómo examinar manualmente el teléfono inteligente, seguido de un enfoque automático usando ADEL. Por último, demostraremos cómo combinar los datos del análisis para crear perfiles de movimiento.

Examen manual

El primer paso es obtener acceso de root al teléfono inteligente. Esto es necesario para eludir las protecciones internas del sistema y obtener acceso a todos los datos. Obtener acceso root es diferente para la mayoría de los teléfonos y depende en gran medida de la versión del sistema operativo. La mejor manera es crear su propia **imagen de recuperación** y arrancar el teléfono a través del modo de recuperación incorporado.

Después de obtener el acceso root, el siguiente paso es intentar que la pantalla se bloquee en texto sin formato ya que este `secret` se usa a menudo para diferentes protecciones (por ejemplo, el bloqueo de pantalla se puede usar como contraseña de una aplicación en el teléfono). Se puede romper el bloqueo de pantalla para un PIN o contraseña con la siguiente secuencia de comandos:

```
import os, sys, subprocess, binascii, struct
import sqlite3 as lite

def get_sha1hash(backup_dir):

    # dumping the password/pin from the device
    print "Dumping PIN/Password hash ..."
    password = subprocess.Popen(['adb', 'pull',
'/data/system/ password.key', backup_dir],
        stdout=subprocess.PIPE, stdin=subprocess.PIPE,
stderr=subprocess.PIPE)
    password.wait()

    # cutting the HASH within password.key
```

```
    sha1hash = open(backup_dir + '/password.key', 'r').readline()[:40]
    print "HASH: \033[0;32m" + sha1hash + "\033[m"

    return sha1hash

def get_salt(backup_dir):

    # dumping the system DB containing the SALT
    print "Dumping locksettings.db ..."
    saltdb = subprocess.Popen(['adb', 'pull', '/data/system/
locksettings.db', backup_dir],
        stdout=subprocess.PIPE, stdin=subprocess.PIPE,
stderr=subprocess.PIPE)
    saltdb.wait()
    saltdb2 = subprocess.Popen(['adb', 'pull', '/data/system/
locksettings.db-wal', backup_dir],
        stdout=subprocess.PIPE, stdin=subprocess.PIPE,
stderr=subprocess.PIPE)
    saltdb2.wait()
    saltdb3 = subprocess.Popen(['adb', 'pull', '/data/system/
locksettings.db-shm', backup_dir],
        stdout=subprocess.PIPE, stdin=subprocess.PIPE,
stderr=subprocess.PIPE)
    saltdb3.wait()

    # extract the SALT
    con = lite.connect(backup_dir + '/locksettings.db')
    cur = con.cursor()
    cur.execute("SELECT value FROM locksettings WHERE
name='lockscreen.password_salt'")
    salt = cur.fetchone()[0]
    con.close()

    # convert SALT to Hex
    returnedsalt =  binascii.hexlify(struct.pack('>q', int(salt) ))
    print "SALT: \033[0;32m" + returnedsalt + "\033[m"

    return returnedsalt


def write_crack(salt, sha1hash, backup_dir):

    crack = open(backup_dir + '/crack.hash', 'a+')

    # write HASH and SALT to cracking file
```

```
        hash_salt = sha1hash + ':' + salt
        crack.write(hash_salt)
        crack.close()


if __name__ == '__main__':

    # check if device is connected and adb is running as root
    if subprocess.Popen(['adb', 'get-state'],
stdout=subprocess.PIPE). communicate(0)[0].split("\n")[0] ==
"unknown":
        print "no device connected - exiting..."
        sys.exit(2)

    # starting to create the output directory and the crack file used
for hashcat
    backup_dir = sys.argv[1]

    try:
        os.stat(backup_dir)
    except:
        os.mkdir(backup_dir)

    sha1hash = get_sha1hash(backup_dir)
    salt = get_salt(backup_dir)
    write_crack(salt, sha1hash, backup_dir)
```

Este script genera un archivo llamado `crack.hash` que se puede usar para alimentar a hashcat para forzar el bloqueo de pantalla. Si el propietario del teléfono inteligente ha utilizado un PIN de 4 dígitos, el comando para ejecutar hashcat es el siguiente:

```
user@lab:~$ ./hashcat -a 3 -m 110 out/crack.hash -1 ?d ?1?1?1?1
Initializing hashcat v0.50 with 4 threads and 32mb segment-
size...

Added hashes from file crack.hash: 1 (1 salts)
Activating quick-digest mode for single-hash with salt

c87226fed37977772be870d722c449f915844922:256c05b54b73308b:0420

All hashes have been recovered

Input.Mode: Mask (?1?1?1?1) [4]
Index.....: 0/1 (segment), 10000 (words), 0 (bytes)
```

```
Recovered.: 1/1 hashes, 1/1 salts
Speed/sec.: - plains, 7.71k words
Progress..: 7744/10000 (77.44%)
Running...: 00:00:00:01
Estimated.: --:--:--:--

Started: Sat Jul 20 17:14:52 2015
Stopped: Sat Jul 20 17:14:53 2015
```

Al mirar la línea marcada en la salida, puede ver el hash sha256 seguido de la salt y el PIN de fuerza bruta que se utiliza para desbloquear la pantalla.

Si el usuario del teléfono inteligente ha usado un gesto para desbloquear el teléfono inteligente, puede usar una tabla rainbow pregenerada y el siguiente script:

```
import hashlib, sqlite3, array, datetime
from binascii import hexlify

SQLITE_DB = "GestureRainbowTable.db"

def crack(backup_dir):

    # dumping the system file containing the hash
    print "Dumping gesture.key ..."

    saltdb = subprocess.Popen(['adb', 'pull',
'/data/system/gesture. key', backup_dir],
        stdout=subprocess.PIPE, stdin=subprocess.PIPE,
stderr=subprocess.PIPE)

    gesturehash = open(backup_dir + "/gesture.key",
    "rb").readline() lookuphash = hexlify(gesturehash).decode()
    print "HASH: \033[0;32m" + lookuphash + "\033[m"

    conn = sqlite3.connect(SQLITE_DB)
    cur = conn.cursor()
    cur.execute("SELECT pattern FROM RainbowTable WHERE hash = ?",
(lookuphash,))
    gesture = cur.fetchone()[0]

    return gesture

if   name__ == '  main  ':

    # check if device is connected and adb is running as root
```

```
    if subprocess.Popen(['adb', 'get-state'],
stdout=subprocess.PIPE). communicate(0)[0].split("\n")[0] ==
"unknown":
        print "no device connected - exiting..."
        sys.exit(2)

    # starting to create the output directory and the crack file used
for hashcat
    backup_dir = sys.argv[1]

    try:
        os.stat(backup_dir)
    except:
        os.mkdir(backup_dir)

    gesture = crack(backup_dir)

    print "screenlock gesture: \033[0;32m" + gesture + "\033[m""
```

Lo siguiente que podría ser muy importante al buscar dispositivos potencialmente infectados es una lista de aplicaciones instaladas y sus hashes para verificarlos contra **AndroTotal** o **Mobile-Sandbox**. Esto se puede hacer con el siguiente script:

```
import os, sys, subprocess, hashlib


def get_apps():

    # dumping the list of installed apps from the device
    print "Dumping apps meta data ..."

    meta = subprocess.Popen(['adb', 'shell', 'ls', '-l',
        '/data/app'], stdout=subprocess.PIPE, stdin=subprocess.PIPE,
stderr=subprocess.PIPE)
    meta.wait()

    apps = []
    while True:
        line = meta.stdout.readline()
        if line != '':
            name = line.split(' ')[-1].rstrip()
            date = line.split(' ')[-3]
            time = line.split(' ')[-2]
            if name.split('.')[-1] == 'apk':
                app = [name, date, time]
            else:
```

```
                continue
        else:
            break
        apps.append(app)

    return apps


def dump_apps(apps, backup_dir):

    # dumping the apps from the device
    print "Dumping the apps ..."

    for app in apps:
        app = app[0]
        subprocess.Popen(['adb', 'pull', '/data/app/' + app, backup_
dir],
            stdout=subprocess.PIPE, stdin=subprocess.PIPE,
stderr=subprocess.PIPE)


def get_hashes(apps, backup_dir):

    # calculating the hashes
    print "Calculating the sha256 hashes ..."

    meta = []
    for app in apps:
        sha256 = hashlib.sha256(open(backup_dir + '/' + app[0], 'rb').
read()).hexdigest()
        app.append(sha256)
        meta.append(app)

    return meta


if   name__ == '  main  ':

    # check if device is connected and adb is running as root
    if subprocess.Popen(['adb', 'get-state'], stdout=subprocess.PIPE).
communicate(0)[0].split("\n")[0] == "unknown":
        print "no device connected - exiting..."
        sys.exit(2)

    # starting to create the output directory
```

```
    backup_dir = sys.argv[1]

    try:
        os.stat(backup_dir)
    except:
        os.mkdir(backup_dir)

    apps = get_apps()
    dump_apps(apps, backup_dir)
    meta = get_hashes(apps, backup_dir)

    # printing the list of installed apps
    print 'Installed apps:'
    for app in meta:
        print "\033[0;32m" + ' '.join(app) + "\033[m"
```

Después de ejecutar el anterior script, obtendrá el siguiente resultado, incluidos los metadatos importantes:

```
user@lab:~$ ./get_installed_apps.py out


Dumping apps meta data ...
Dumping the apps ...
Calculating the sha256 hashes ...


Installed apps:
com.android.SSLTrustKiller-1.apk 2015-05-18 17:11
52b4d6a1888a6514b62f6607cebf8c2c2aa4e4857319ec67b24be601db5243fb
com.android.chrome-2.apk 2015-06-16 20:50
191cd720626df38eaedf3301826e72330493cdeb8c45da4e309939cfe5633d61
com.android.vending-1.apk 2015-07-25 12:05
7be9f8f99e8c1a6c3be1edb01d84aba14619e3c67c14856755523413ba8e2d98
com.google.android.GoogleCamera-2.apk 2015-06-16 20:49
6936f3c17948c767550c206ff0ae0f44f1f4da0fcb85125da722e0c709787894
com.google.android.apps.authenticator2-1.apk 2015-06-05 10:14
11bcfcf1c853b1eb567c9453507c3413b09a1d70fd3085013f4a091719560ab6
...
```

Con la ayuda de esta información, puede verificar las aplicaciones contra los servicios en línea para saber si son seguros de usar o potencialmente maliciosos. Si no desea enviarlos, puede usar el `script apk_analyzer.py` en combinación con **Androguard** para realizar un análisis rápido que a menudo puede revelar información importante.

Después de obtener una lista de todas las aplicaciones instaladas y verificar que no tengan un comportamiento malicioso, también puede ser muy útil obtener información sobre todas las particiones y puntos de montaje del dispositivo. Esto se puede lograr con el siguiente script:

```
import sys, subprocess


def get_partition_info():

    # dumping the list of installed apps from the device
    print "Dumping partition information ..."

    partitions = subprocess.Popen(['adb', 'shell', 'mount'],
        stdout=subprocess.PIPE, stdin=subprocess.PIPE,
stderr=subprocess.PIPE)
    partitions.wait()

    while True:
        line = partitions.stdout.readline().rstrip()
        if line != '':
            print "\033[0;32m" + line + "\033[m"
        else:
            break


if   name__ == '  main  ':

    # check if device is connected and adb is running as root
    if subprocess.Popen(['adb', 'get-state'],
stdout=subprocess.PIPE). communicate(0)[0].split("\n")[0] ==
"unknown":
        print "no device connected - exiting..."
        sys.exit(2)

    get_partition_info()
```

La salida de un teléfono rooteado podría verse así:

```
user@lab:~$ ./get_partitions.py


Dumping partition information ...
rootfs / rootfs rw,relatime 0 0
tmpfs /dev tmpfs rw,seclabel,nosuid,relatime,mode=755 0 0
devpts /dev/pts devpts rw,seclabel,relatime,mode=600 0 0
proc /proc proc rw,relatime 0 0
sysfs /sys sysfs rw,seclabel,relatime 0 0
selinuxfs /sys/fs/selinux selinuxfs rw,relatime 0 0
debugfs /sys/kernel/debug debugfs rw,relatime 0 0
none /acct cgroup rw,relatime,cpuacct 0 0
none /sys/fs/cgroup tmpfs rw,seclabel,relatime,mode=750,gid=1000 0 0
tmpfs /mnt/asec tmpfs rw,seclabel,relatime,mode=755,gid=1000 0 0
tmpfs /mnt/obb tmpfs rw,seclabel,relatime,mode=755,gid=1000 0 0
none /dev/cpuctl cgroup rw,relatime,cpu 0 0
/dev/block/platform/msm_sdcc.1/by-name/system /system ext4
ro,seclabel,re latime,data=ordered 0 0
/dev/block/platform/msm_sdcc.1/by-name/userdata /data ext4 rw
,seclabel,nosuid,nodev,noatime,nomblk_io_submit,noauto_da_
alloc,errors=panic,data=ordered 0 0
/dev/block/platform/msm_sdcc.1/by-name/cache /cache ext4 rw
,seclabel,nosuid,nodev,noatime,nomblk_io_submit,noauto_da_
alloc,errors=panic,data=ordered 0 0
/dev/block/platform/msm_sdcc.1/by-name/persist       /persist       ext4
rw,seclabel,
nosuid,nodev,relatime,nomblk_io_submit,nodelalloc,errors=panic,data=orde
r ed 0 0
/dev/block/platform/msm_sdcc.1/by-name/modem /firmware vfat
ro,relatime,u
id=1000,gid=1000,fmask=0337,dmask=0227,codepage=cp437,iocharset=iso8859-
1,shortname=lower,errors=remount-ro 0 0
/dev/fuse /mnt/shell/emulated fuse
rw,nosuid,nodev,relatime,user_
id=1023,group_id=1023,default_permissions,allow_other 0 0
```

Al final de esta sección, le mostraremos cómo obtener más detalles sobre el uso del teléfono inteligente basado en Android. En el siguiente ejemplo, usaremos la base de datos de contactos que también almacena el historial de llamadas telefónicas. Este ejemplo se puede adoptar fácilmente para obtener entradas de calendario o contenido de cualquier otra base de datos de una aplicación que esté instalada en el dispositivo:

```
import os, sys, subprocess
import sqlite3 as lite
from prettytable import from_db_cursor


def dump_database(backup_dir):

    # dumping the password/pin from the device
    print "Dumping contacts database ..."

    contactsDB = subprocess.Popen(['adb', 'pull',
'/data/data/com.
android.providers.contacts/databases/contacts2.db',
        backup_dir], stdout=subprocess.PIPE,
stdin=subprocess.PIPE, stderr=subprocess.PIPE)
    contactsDB.wait()


def get_content(backup_dir):

       # getting the content from the contacts database
       con = lite.connect(backup_dir + '/contacts2.db')
       cur = con.cursor()
       cur.execute("SELECT contacts._id AS _id,contacts.custom_ringtone
  AS custom_ringtone, name_raw_contact.display_name_source AS display_
  name_source, name_raw_contact.display_name AS display_name, name_
  raw_contact.display_name_alt AS display_name_alt, name_raw_contact.
  phonetic_name AS phonetic_name, name_raw_contact.phonetic_name_style
  AS phonetic_name_style, name_raw_contact.sort_key AS sort_key, name_
  raw_contact.phonebook_label AS phonebook_label, name_raw_contact.
  phonebook_bucket AS phonebook_bucket, name_raw_contact.sort_key_alt
  AS sort_key_alt, name_raw_contact.phonebook_label_alt AS phonebook_
  label_alt, name_raw_contact.phonebook_bucket_alt AS phonebook_
  bucket_alt, has_phone_number, name_raw_contact_id, lookup, photo_id,
  photo_file_id, CAST(EXISTS (SELECT _id FROM visible_contacts WHERE
  contacts._id=visible_contacts._id) AS INTEGER) AS in_visible_group,
  status_update_id, contacts.contact_last_updated_timestamp, contacts.
  last_time_contacted AS last_time_contacted, contacts.send_to_voicemail
  AS send_to_voicemail, contacts.starred AS starred, contacts.pinned
  AS pinned, contacts.times_contacted AS times_contacted, (CASE WHEN
  photo_file_id IS NULL THEN (CASE WHEN photo_id IS NULL OR photo_id=0
  THEN NULL ELSE 'content://com.android.contacts/contacts/'||contacts._
```

```
id|| '/photo' END) ELSE 'content://com.android.contacts/display_
photo/'||photo_file_id END) AS photo_uri, (CASE WHEN photo_id IS
NULL OR photo_id=0 THEN NULL ELSE 'content://com.android.contacts/
contacts/'||contacts._id|| '/photo' END) AS photo_thumb_uri, 0 AS
is_user_profile FROM contacts JOIN raw_contacts AS name_raw_contact
ON(name_raw_contact_id=name_raw_contact._id)")
    pt = from_db_cursor(cur)
    con.close()

    print pt


if __name__ == '__main__':

    # check if device is connected and adb is running as root
    if subprocess.Popen(['adb', 'get-state'], stdout=subprocess.PIPE).
communicate(0)[0].split("\n")[0] == "unknown":
        print "no device connected - exiting..."
        sys.exit(2)

    # starting to create the output directory
    backup_dir = sys.argv[1]

    try:
        os.stat(backup_dir)
    except:
        os.mkdir(backup_dir)

    dump_database(backup_dir)
    get_content(backup_dir)
```

Después de haber visto cómo realizar manualmente un análisis de un teléfono inteligente, le mostraremos, en la próxima sección, cómo realizar las mismas acciones que están automatizadas con la ayuda de ADEL.

Examen automatizado con la ayuda de ADEL

Hemos desarrollado una herramienta llamada ADEL. Inicialmente se desarrolló para las versiones 2.x de Android, pero se actualizó para adaptarse a las necesidades de análisis de teléfonos inteligentes con Android 4.x. Esta herramienta puede volcar automáticamente los archivos de base de datos SQLite seleccionados de dispositivos Android y extraer los contenidos que están almacenados en los archivos vacíos. Como una opción adicional, ADEL puede analizar bases de datos que se descargaron manualmente de antemano. Esta opción se implementó para admitir teléfonos inteligentes en los que ADEL no puede acceder al sistema de archivos del dispositivo debido a características de seguridad como los cargadores de arranque bloqueados. En las siguientes secciones, describimos las tareas principales de ADEL y los pasos que la herramienta realmente realiza.

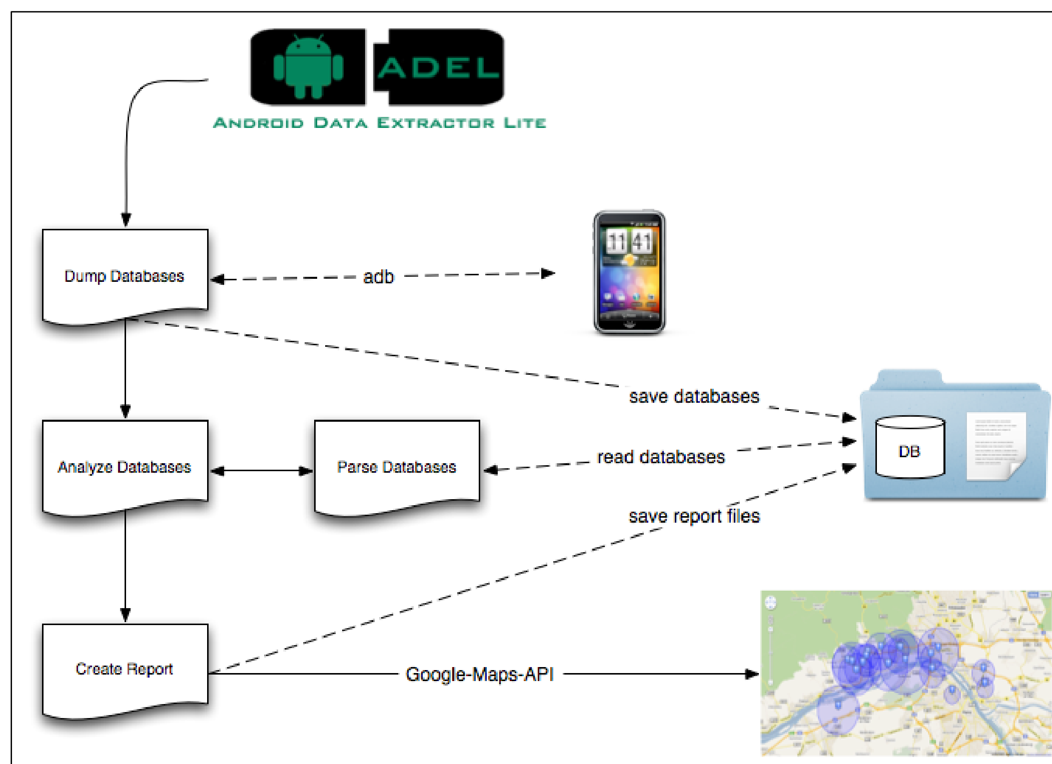
Idea detrás del sistema

Durante el desarrollo de ADEL, tomamos en cuenta principalmente las siguientes pautas de diseño:

- **Principios forenses**: ADEL intenta tratar los datos de una manera forense correcta. Este objetivo se logra por el hecho de que las actividades no se realizan directamente en el teléfono, sino en una copia de las bases de datos. Este procedimiento asegura que los datos no sean modificados por los usuarios de Adel o por un sistema operativo comprometido. Para proporcionar la prueba de la corrección forense de ADEL, los valores hash se calculan antes y después de cada análisis para garantizar que los datos objeto de dumping no se hayan modificado durante el análisis.
- **Extensibilidad:** ADEL ha sido construido modularmente y contiene dos módulos separados: el análisis y el módulo de informe. Existen interfaces predefinidas entre estos módulos y ambos se pueden modificar fácilmente con la ayuda de funciones adicionales. La estructura modular le permite volcar y analizar otras bases de datos de teléfonos inteligentes sin grandes esfuerzos y facilita las actualizaciones del sistema en el futuro.
- **Usabilidad:** el uso de ADEL pretende ser lo más simple posible para permitir su uso tanto por personas calificadas como no expertas. En el mejor de los casos, el análisis del teléfono móvil se realiza de forma autónoma para que el usuario no reciba ninguna notificación de los procesos internos. Además, el módulo de informe crea un informe detallado en una forma legible que incluye todos los datos decodificados. Durante la ejecución, ADEL escribe opcionalmente un extenso archivo de registro donde se rastrean todos los pasos importantes que se ejecutaron.

Implementación y flujo de trabajo del sistema

Un diagrama de flujo que muestra la estructura de ADEL se muestra en la siguiente figura:



ADEL utiliza el Kit de desarrollo de software de Android (SDK de Android) para volcar los archivos de la base de datos en la máquina del investigador. Para extraer contenidos que están contenidos en un archivo de base de datos SQLite, ADEL analiza las estructuras de datos de bajo nivel. Después de abrir el archivo de base de datos que se analizará en el modo de solo lectura, ADEL lee el encabezado de la base de datos (los primeros 100 bytes del archivo) y extrae los valores para cada uno de los campos del encabezado. No todos, pero algunos de los valores en los campos del encabezado son necesarios para analizar el resto del archivo de la base de datos. Un valor importante es el tamaño de las páginas en el archivo de la base de datos, que es necesario para analizar las estructuras del árbol B (por página). Después de haber leído los campos del encabezado de la base de datos, ADEL analiza el árbol B que contiene la tabla `sqlite_master` para la cual la primera página de la base de datos siempre es la página raíz. La declaración SQL CREATE y el número de página de la página raíz del árbol B se extraen para cada una de las tablas de la base de datos. Además, la declaración SQL CREATE se analiza adicionalmente para extraer el nombre y el tipo de datos de cada columna de la tabla correspondiente.

Finalmente, la estructura completa del árbol B se analiza para cada tabla, comenzando en la página raíz del árbol B, que se extrajo de la tabla sqlite_master. Al seguir los punteros de todas las páginas interiores, puede identificar cada página de hoja del árbol B. Finalmente, el contenido de la fila de cada tabla se extrae de las celdas que se encuentran en cualquier hoja de página que pertenece a la misma tabla del árbol B.

En las siguientes secciones, abordaremos el módulo de informe y sus funcionalidades. En el estado actual de desarrollo, las siguientes bases de datos son tratadas y analizadas de manera forense:

- Información de teléfono y tarjeta SIM (por ejemplo, I**dentidad internacional del suscriptor móvil (IMSI)** y número de serie)
- Agenda telefónica y listas de llamadas
- Entradas de calendario
- Mensajes SMS
- Mapas de Google

Los datos recuperados de esta manera se escriben en un archivo XML por el módulo de informes para facilitar el uso y la representación de los datos. De forma similar al módulo de análisis, se puede actualizar fácilmente con respecto a posibles cambios en futuras versiones de Android o en esquemas de bases de datos subyacentes. Por lo tanto, hemos creado diferentes tuplas, por ejemplo, [tabla, fila, columna], para definir los datos intercambiados entre ambos módulos. Si el diseño de la base de datos cambia en el futuro, solo la tupla debe adaptarse. El módulo de informe crea automáticamente archivos XML para cada tipo de datos que se enumera previamente. Además, se crea un informe que contiene todos los datos extraídos de las bases de datos analizadas. Con la ayuda de un archivo XSL, el informe será reformado gráficamente. Todos los archivos creados por ADEL se almacenan en una subcarpeta del proyecto actual.

Para acceder a las bases de datos y las carpetas del sistema necesarias en el teléfono inteligente, ADEL necesita acceso root en el dispositivo.

Trabajar con ADEL

Después de haber descrito qué es ADEL y cómo funciona, ahora vamos a ir a la parte práctica de esta sección y comenzar a usarlo. Puede descargar ADEL desde la siguiente URL: https://mspreitz.github.io/ADEL

Todo lo que necesita hacer es verificar si el dispositivo en cuestión ya está incluido en el perfil de configuración de ADEL que se encuentra en **/xml/phone_config.xml.** Si falta el dispositivo, hay dos opciones sobre cómo proceder:

1. Elija un dispositivo diferente con la misma versión de Android (esto generará una advertencia, pero funciona en la mayoría de los casos).
2. Genere una nueva configuración de dispositivo que coincida con el tipo de dispositivo y la versión de Android del dispositivo en cuestión.

Si elige la segunda opción, puede copiar la configuración de un dispositivo que ya está funcionando y adoptar los números en el archivo XML. Estos números representan las tablas y columnas de la base de datos indicada. Para ser un poco más preciso, si intenta adoptar la base de datos de SMS, debe verificar los números de las siguientes tablas y columnas:

```
<sms>
  <db_name>mmssms.db</db_name>
  <table_num>10</table_num>
  <sms_entry_positions>
    <id>0</id>
    <thread_id>1</thread_id>
    <address>2</address>
    <person>3</person>
    <date>4</date>
    <read>7</read>
    <type>9</type>
    <subject>11</subject>
    <body>12</body>
  </sms_entry_positions>
</sms>
```

El número para la etiqueta `table_num` debe establecerse en el número que corresponde a la tabla llamada `sms`. Deben adoptarse los siguientes números correspondientes a las columnas en la tabla `sms` que se nombran de manera idéntica. El ejemplo impreso anterior funciona con un Nexus 5 y Android 4.4.4. Lo mismo tiene que hacerse para todas las demás bases de datos también.

Ejecutar ADEL contra un Nexus 5 rooteado con Android 4.4.4 lleno de datos de prueba, genera el siguiente resultado:

```
user@lab:~$./adel.py -d nexus5 -l 4

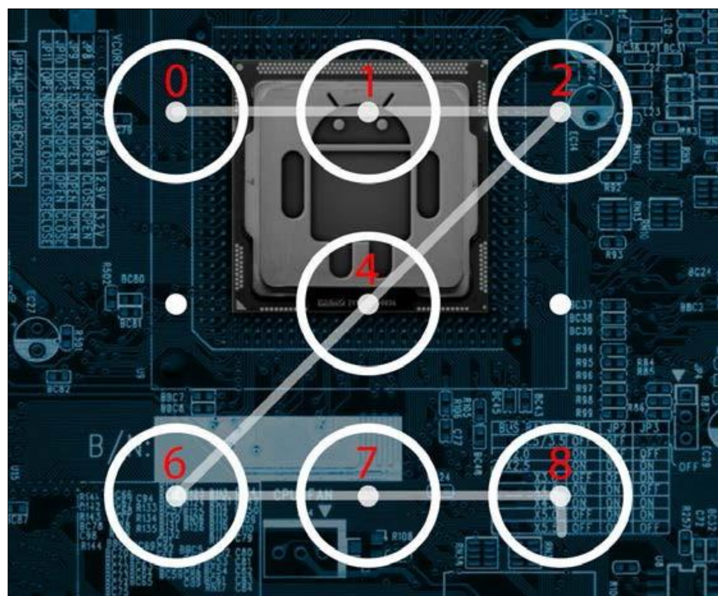

              _____  ________  ___________.____
             /  _  \ \______ \ \_   _____/|    |
            /  /_\  \ |    |  \ |    __)_ |    |
           /    |    \|    `   \|        \|    |___
           \____|__  /_______  /_______  /|_______ \
                   \/        \/        \/         \/
               Android Data Extractor Lite v3.0


ADEL MAIN:     ----> starting script....
ADEL MAIN:     ----> Trying to connect to smartphone or emulator....
dumpDBs:       ----> opening connection to device: 031c6277f0a6a117
dumpDBs:       ----> evidence directory 2015-07-20  22-53-22  031c6277f0a
6a117 created
ADEL MAIN:     ----> log file 2015-07-20  22-53-22  031c6277f0a6a117/log/
adel.log created
ADEL MAIN:     ----> log level: 4
dumpDBs:       ----> device is running Android OS 4.4.4
dumpDBs:       ----> dumping all SQLite databases....
dumpDBs:       ----> auto dict doesn't exist!
dumpDBs:       ----> weather database doesn't exist!
dumpDBs:       ----> weather widget doesn't exist!
dumpDBs:       ----> Google-Maps navigation history doesn't exist!
dumpDBs:       ----> Facebook database doesn't exist!
dumpDBs:       ----> Cached geopositions within browser don't exist!
dumpDBs:       ----> dumping pictures (internal_sdcard)....
dumpDBs:       ----> dumping pictures (external_sdcard)....
dumpDBs:       ----> dumping screen captures (internal_sdcard)....
dumpDBs:       ----> dumping screen captures (internal_sdcard)....
dumpDBs:       ----> all SQLite databases dumped
Screenlock:    ----> Screenlock Hash:
6a062b9b3452e366407181a1bf92ea73e9ed4c48
```

```
Screenlock:    ----> Screenlock Gesture: [0, 1, 2, 4, 6, 7, 8]
LocationInfo:  ----> Location map 2015-07-20  22-53-22  031c6277f0a6a117/
map.html created
analyzeDBs:    ----> starting to parse and analyze the databases....
parseDBs:      ----> starting to parse smartphone info
parseDBs:      ----> starting to parse calendar entries
parseDBs:      ----> starting to parse SMS messages
parseDBs:      ----> starting to parse call logs
parseDBs:      ----> starting to parse address book entries
analyzeDBs:    ----> all databases parsed and analyzed....
createReport:  ----> creating report....
ADEL MAIN:     ----> report 2015-07-20  22-53-22  031c6277f0a6a117/xml/
report.xml created
compareHash:   ----> starting to compare calculated hash values
ADEL MAIN:     ----> stopping script....

         (c) m.spreitzenbarth & s.schmitt 2015
```

En esta salida, puede ver el nombre de la carpeta donde se descargan todos los datos y dónde se puede encontrar el informe generado. Además, también puede ver el gesto del bloqueo de pantalla que se extrajo automáticamente y se comparó con una tabla rainbow pregenerada, de la siguiente manera:



Perfiles de movimiento

Además de los datos sobre las comunicaciones individuales, la directiva de la UE de 2006 también exige que los operadores de red retengan ciertos datos de ubicación. Especialmente, la directiva exige que los siguientes datos se conserven durante al menos seis meses:

- Identidad y coordenadas GPS exactas de la celda de radio donde el usuario inició una llamada telefónica
- Identidad y coordenadas de la celda de radio que estaba activa al comienzo de una transmisión de datos GPRS
- Sellos de tiempo correspondientes a esta información

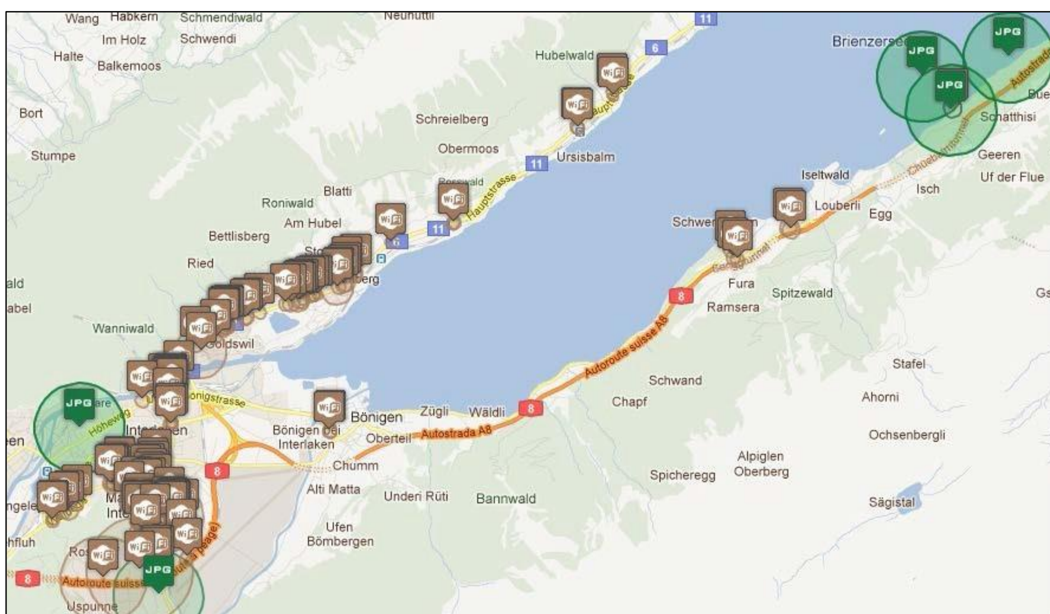
Esta información puede ayudar a los investigadores a crear perfiles de movimiento de sospechosos. Además, la información se puede usar para ubicar y monitorear sospechosos.

Muchos países miembros de la UE han implementado esta directiva en las leyes nacionales. Sin embargo, en algunos países, ha habido un intenso debate público sobre las leyes, especialmente en relación con las amenazas a la privacidad. En Alemania, las discusiones fueron alimentadas por un conjunto de datos proporcionados por el político alemán Malte Spitz. El conjunto de datos contenía datos de su ubicación durante un período de seis meses que fue preservado por su operador de red móvil según la ley de retención de datos. Un periódico alemán creó una interfaz gráfica que permitía a los usuarios reproducir visualmente los movimientos detallados de Spitz.

En general, se argumenta que retener grandes cantidades de datos crea nuevos riesgos de abuso. Además, el requisito de almacenar datos pertenecientes a millones de personas inocentes es desproporcionado con respecto al pequeño número de casos en que los datos son utilizados por las autoridades. Como resultado, en 2011, el Tribunal Constitucional alemán desestimó la legislación original que requería retención de datos. Mientras tanto, la búsqueda de técnicas menos invasivas para analizar los movimientos de delincuentes continúa.

En los últimos años, muchos tipos nuevos de teléfonos móviles (teléfonos inteligentes) han inundado el mercado. Como son esencialmente computadoras personales pequeñas, ofrecen mucho más que la posibilidad de hacer llamadas telefónicas y navegar por Internet. Cada vez más suscriptores usan aplicaciones (principalmente aplicaciones de terceros que se instalan directamente en sus teléfonos) y se comunican con amigos y familiares a través de las redes sociales como Facebook, Google+ y Twitter.

Por rendimiento y otras razones, los dispositivos móviles almacenan de forma persistente los datos de ubicación en la memoria local. En abril de 2011, se informó que Android e iOS almacenan datos geográficos confidenciales. Estos datos, que se mantienen en los archivos de caché del sistema, se envían regularmente a los desarrolladores de la plataforma. Sin embargo, la generación de datos geográficos no está restringida al sistema operativo: muchas aplicaciones que brindan servicios basados en la ubicación también crean y almacenan dichos datos. Por ejemplo, Benford ha demostrado que las imágenes tomadas por un iPhone contienen las coordenadas GPS de la ubicación donde se tomaron las imágenes. Dichos datos son sensibles porque se pueden usar para crear perfiles de movimiento como se ve en la siguiente figura. A diferencia de los datos de ubicación que conservan los operadores de red, la aplicación de la ley puede acceder a los datos de ubicación almacenados en teléfonos inteligentes a través de un decomiso abierto.



Apple iOS

Después de haber visto cómo examinar un teléfono inteligente basado en Android, ahora queremos mostrarle cómo realizar investigaciones similares en dispositivos basados en iOS. En la primera sección, estamos utilizando una conexión **Secure Shell (SSH)** para el dispositivo y le mostraremos cómo obtener los datos almacenados de keychain de un dispositivo iOS con jailbreak.

En la segunda parte de esta sección, usaremos **libimobiledevice.** Esta biblioteca es una biblioteca multiplataforma que utiliza los protocolos para admitir dispositivos basados en iOS y le permite acceder fácilmente al sistema de archivos del dispositivo, recuperar información sobre el dispositivo y sus funciones internas, realizar copias de seguridad/restauracion del dispositivo, administrar aplicaciones instaladas, recuperar datos PIM así como también marcadores, etc. El hecho más importante es que el dispositivo basado en iOS no tiene que ser jailbroken para ser utilizado, cuando se trata de libimobiledevice.

Conseguir el Keychain de un jailbroken iDevice

En muchos casos, puede ser muy útil obtener nombres de usuario y contraseñas de cuentas que el usuario del iDevice estaba usando. Este tipo de datos se encuentra en el keychain de iOS y se puede extraer de iDevice con la ayuda del siguiente script:

```
import os, sys, subprocess


def get_kc(ip, backup_dir):

    # dumping the keychain
    print "Dumping the keychain ..."

    kc = subprocess.Popen(['scp', 'root@' + ip +
':/private/var/ Keychains/keychain-2.db', backup_dir],
        stdout=subprocess.PIPE, stdin=subprocess.PIPE,
stderr=subprocess.PIPE)
    kc.communicate()


def push_kcd(ip):

    # dumping the keychain
    print "Pushing the Keychain Dumper to the device ..."

    kcd = subprocess.Popen(['scp', 'keychain_dumper' 'root@' + ip +
':~/'],
        stdout=subprocess.PIPE, stdin=subprocess.PIPE,
stderr=subprocess.PIPE)
    kcd.communicate()


def exec_kcd(ip, backup_dir):
```

```
    # pretty print keychain
    kcc = subprocess.Popen(['ssh', 'root@' + ip, './keychain_dumper'],
        stdout=subprocess.PIPE, stdin=subprocess.PIPE,
stderr=subprocess.PIPE)
    kcc.communicate()
    kcc.stdout


if   name__ == '  main  ':

    # starting to create the output directory
    backup_dir = sys.argv[1]

    try:
        os.stat(backup_dir)
    except:
        os.mkdir(backup_dir)

    # get the IP of the iDevice from user input
    ip = sys.argv[2]

    get_kc(ip, backup_dir)
    push_kcd(ip)
    exec_kcd(ip, backup_dir)
```

En el resultado del script anterior, también puede encontrar la contraseña de la cuenta de Apple en la que está registrado el dispositivo:

```
Generic Password
----------------
Service: com.apple.account.AppleAccount.password
Account: 437C2D8F-****-****-****-************
Entitlement Group: apple
Label: (null)
Generic Field: (null)
Keychain Data: ************************
```

Examen manual con libimobiledevice

Esta biblioteca utiliza protocolos iOS comunes para la comunicación entre la máquina del investigador y el iDevice conectado. Para funcionar correctamente, el dispositivo debe estar desbloqueado y emparejado porque, de lo contrario, una gran cantidad de datos en el dispositivo todavía está encriptados y, por lo tanto, protegidos.

Con la ayuda del siguiente script, puede crear una copia de seguridad completa del dispositivo (similar a una copia de seguridad de iTunes). Posteriormente, el script descomprimirá la copia de seguridad e imprimirá una lista jerárquica de todos los archivos y carpetas en la copia de seguridad. Dependiendo del tamaño del iDevice, este script puede ejecutarse durante varios minutos.

```
import os, sys, subprocess


def get_device_info():

    # getting the udid of the connected device
    udid = subprocess.Popen(['idevice_id', '-l'], stdout=subprocess.
PIPE).stdout.readline().rstrip()

    print "connected device: \033[0;32m" + udid + "\033[m"
    return udid


def create_backup(backup_dir):

    # creating a backup of the connected device
    print "creating backup (this can take some time) ..."

    backup = subprocess.Popen(['idevicebackup2', 'backup',
backup_ dir], stdout=subprocess.PIPE)
    backup.communicate()

    print "backup successfully created in ./" + backup_dir + "/"


def unback_backup(udid, backup_dir):

    # unpacking the backup
    print "unpacking the backup ..."

    backup = subprocess.Popen(['idevicebackup2', '-u', udid,
'unback', backup_dir], stdout=subprocess.PIPE)
```

```
    backup.communicate()

    print "backup successfully unpacked and ready for analysis"


def get_content(backup_dir):

    # printing content of the created backup
    content = subprocess.Popen(['tree', backup_dir + '/_unback_/'],
stdout=subprocess.PIPE).stdout.read()
    f = open(backup_dir + '/filelist.txt', 'a+')
    f.write(content)
    f.close

    print "list of all files and folders of the backup are stored in
./" + backup_dir + "/filelist.txt"


if __name__ == '__main__':

    # check if device is connected
    if subprocess.Popen(['idevice_id', '-l'], stdout=subprocess.PIPE).
communicate(0)[0].split("\n")[0] == "":
        print "no device connected - exiting..."
        sys.exit(2)

    # starting to create the output directory
    backup_dir = sys.argv[1]

    try:
        os.stat(backup_dir)
    except:
        os.mkdir(backup_dir)

    udid = get_device_info()
    create_backup(backup_dir)
    unback_backup(udid, backup_dir)
    get_content(backup_dir)
```

El resultado final de este script se verá como el siguiente extracto:

```
user@lab:~$ ./create_ios_backup.py out


connected device:
460683e351a265a7b9ea184b2802cf4fcd02526d creating backup
(this can take some time) ...
backup successfully created in ./out
unpacking the backup ...
backup successfully unpacked and ready for analysis
list of all files and folders of the backup are stored in
./out/filelist. txt
```

Con la ayuda de la lista de archivos y carpetas, puede comenzar el análisis de la copia de seguridad con herramientas comunes como un visor de archivos plist o un navegador SQLite. La búsqueda de **Cydia App Store** en este archivo generado también puede ayudar a identificar si el usuario o un atacante ha liberado el teléfono inteligente.

Resumen

En este capítulo, cubrimos el modelo de proceso de investigación de Eoghan Casey y lo adoptamos para el caso de los teléfonos inteligentes. Más tarde, realizamos un análisis de los teléfonos inteligentes Android de forma manual y automática con la ayuda de scripts de Python y el framework ADEL. En la última sección, cubrimos el análisis de teléfonos inteligentes basados en iOS.

Después de manejar la investigación forense de los teléfonos inteligentes, terminamos la adquisición y el análisis físico y virtual y cambiaremos la investigación al área volátil de los dispositivos en el siguiente capítulo.

7

El uso de Python para el análisis forense de memoria

Ahora que ha realizado investigaciones en la infraestructura (consulte el Capítulo 4, Uso de Python para análisis forense de redes), equipos de IT comunes (consulte el Capítulo 3, Uso de Python para Windows y Linux Forensics), e incluso en el virtualizado (consulte el Capítulo 5 , Uso de Python para virtualización Forense) y mundos móviles (consulte el Capítulo 6, Uso de Python para forense móvil), en este capítulo, le mostraremos cómo investigar en memoria volátil con la ayuda de Volatility, un framework basado en Python. en las siguientes plataformas:

- Android
- Linux

Después de mostrarle algunos complementos básicos de Volatility para Android y Linux y cómo obtener el volcado de RAM requerido para el análisis, buscaremos malware en la memoria RAM. Por lo tanto, usaremos reglas de YARA basadas en la coincidencia de patrones y las combinaremos con el poder de Volatility.

Comprender los fundamentos de Volatility

En general, la memoria forense sigue el mismo patrón que otras investigaciones forenses:

1. Seleccionando el objetivo de la investigación.
2. Adquirir datos forenses.
3. Análisis forense.

En los capítulos anteriores, ya presentamos varias tecnologías sobre cómo seleccionar el objetivo de una investigación, por ejemplo, comenzando desde el sistema con configuraciones inusuales en la capa de virtualización.

La adquisición de datos forenses para el análisis de la memoria depende en gran medida del entorno y lo discutiremos en las secciones `Uso de Volatility en Linux` y `Uso de Volatility en Android` de este capítulo.



Considere siempre la capa de virtualización como fuente de datos

La adquisición de memoria de un sistema operativo en ejecución siempre requiere acceso administrativo a este sistema y es un proceso intrusivo, es decir, el proceso de adquisición de datos cambia los datos de la memoria. Además, el malware avanzado es capaz de manipular la gestión de la memoria del sistema operativo para evitar su adquisición. Por lo tanto, siempre verifique e intente adquirir la memoria en la capa del hipervisor como se describe en el Capítulo 5, Uso de Python para virtualización Forense

La herramienta más importante para el análisis de datos de memoria es **Volatility**. Volatility está disponible en Volatility Foundation en `http://www.volatilityfoundation.org/`.

La herramienta está escrita en Python y se puede utilizar de forma gratuita bajo los términos de la **Licencia pública general de GNU (GPL)** versión 2. Volatility puede leer volcados de memoria en varios formatos de archivo, por ejemplo, archivos de hibernación, volcados de memoria en bruto, Los archivos de snapshot de memoria de VMware y el formato de **extractor de memoria de Linux (LiME)** producido por el módulo LiME, que se analizarán más adelante en este capítulo.

Los términos más importantes en el mundo de Volatility son los siguientes:

- **Profile**: Un perfil ayuda a Volatility en la interpretación de los desplazamientos de memoria y las estructuras de la memoria. El perfil depende del sistema operativo, especialmente del núcleo del sistema operativo, la máquina y la arquitectura de la CPU. Volatility contiene una variedad de perfiles para los casos de uso más comunes. En la sección Uso de Volatility en Linux de este capítulo, describiremos cómo crear sus perfiles.
- **Plugin**: Los Plugins se utilizan para realizar acciones en el volcado de memoria. Cada comando de Volatility que use llama a un Plugin para realizar la acción correspondiente. Por ejemplo, para obtener una lista de todos los procesos que se estaban ejecutando durante el volcado de memoria de un sistema Linux, se usa el Plugin linux_pslist.

Volatility proporciona una documentación completa y le recomendamos que se familiarice con todas las descripciones de los módulos para aprovechar al máximo Volatility.

Usando Volatility en Android

Para analizar la memoria volátil desde dispositivos Android, primero necesitará LiME. LiME es un **Módulo de kernel cargable (LKM)** que da acceso a toda la memoria RAM del dispositivo y puede descargarlo en una tarjeta SD física o desde la red. Después de adquirir el volcado de memoria volátil con LiME, le mostraremos cómo instalar y configurar Volatility para analizar el volcado de memoria RAM. En la última sección, demostraremos cómo obtener información específica del volcado de memoria RAM.

LiME y la imagen de recuperación

LiME es un Módulo de kernel cargable (LKM) que permite la adquisición de memoria volátil desde Linux y dispositivos basados en Linux, como Android. Esto hace que LiME sea único, ya que es la primera herramienta que permite capturas de memoria completas en dispositivos Android. También minimiza su interacción entre el usuario y los procesos del espacio del kernel durante la adquisición, lo que le permite producir capturas de memoria que son más forense que las de otras herramientas diseñadas para la adquisición de memoria de Linux.

Para utilizar LiME en Android, tiene que compilarse de forma cruzada para el kernel usado en el dispositivo en cuestión. En las siguientes secciones, veremos cómo se realizan estos pasos para un Nexus 4 con Android 4.4.4 (sin embargo, este enfoque se puede adaptar a todos los dispositivos basados en Android para los que el kernel, o al menos la configuración del kernel, está disponible como código abierto).

En primer lugar, tenemos que instalar algunos paquetes adicionales en nuestro sistema de laboratorio, de la siguiente manera:

```
user@lab:~$ sudo apt-get install bison g++-multilib git gperf libxml2-
utils make python-networkx zlib1g-dev:i386 zip openjdk-7-jdk
```

Después de instalar todos los paquetes requeridos, ahora debemos configurar el acceso a los dispositivos USB. Bajo los sistemas GNU/Linux, los usuarios regulares directamente no pueden acceder a los dispositivos USB por defecto. El sistema necesita ser configurado para permitir tal acceso. Esto se hace creando un archivo llamado /etc/udev/rules.d/51-android.rules (como el usuario root) e insertando las siguientes líneas en él:

```
# adb protocol on passion (Nexus One)
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1",
ATTR{idProduct}=="4e12", MODE="0600", OWNER="user"
# fastboot protocol on passion (Nexus One)
SUBSYSTEM=="usb", ATTR{idVendor}=="0bb4",
ATTR{idProduct}=="0fff", MODE="0600", OWNER="user"
# adb protocol on crespo/crespo4g (Nexus S)
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1",
ATTR{idProduct}=="4e22", MODE="0600", OWNER="user"
# fastboot protocol on crespo/crespo4g (Nexus S)
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="4e20",
MODE="0600", OWNER="user"
# adb protocol on stingray/wingray (Xoom)
SUBSYSTEM=="usb", ATTR{idVendor}=="22b8", ATTR{idProduct}=="70a9",
MODE="0600", OWNER="user"
# fastboot protocol on stingray/wingray (Xoom)
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="708c",
MODE="0600", OWNER="user"
# adb protocol on maguro/toro (Galaxy Nexus)
SUBSYSTEM=="usb", ATTR{idVendor}=="04e8", ATTR{idProduct}=="6860",
MODE="0600", OWNER="user"
# fastboot protocol on maguro/toro (Galaxy Nexus)
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="4e30",
MODE="0600", OWNER="user"
# adb protocol on panda (PandaBoard)
SUBSYSTEM=="usb", ATTR{idVendor}=="0451", ATTR{idProduct}=="d101",
MODE="0600", OWNER="user"
# adb protocol on panda (PandaBoard ES)
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="d002",
MODE="0600", OWNER="user"
# fastboot protocol on panda (PandaBoard)
SUBSYSTEM=="usb", ATTR{idVendor}=="0451", ATTR{idProduct}=="d022",
MODE="0600", OWNER="user"
# usbboot protocol on panda (PandaBoard)
SUBSYSTEM=="usb", ATTR{idVendor}=="0451", ATTR{idProduct}=="d00f",
MODE="0600", OWNER="user"
# usbboot protocol on panda (PandaBoard ES)
SUBSYSTEM=="usb", ATTR{idVendor}=="0451", ATTR{idProduct}=="d010",
MODE="0600", OWNER="user"
# adb protocol on grouper/tilapia (Nexus 7)
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="4e42",
MODE="0600", OWNER="user"
# fastboot protocol on grouper/tilapia (Nexus 7)
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="4e40",
MODE="0600", OWNER="user"
# adb protocol on manta (Nexus 10)
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="4ee2",
MODE="0600", OWNER="user"
# fastboot protocol on manta (Nexus 10)
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="4ee0",
MODE="0600", OWNER="user"
```

Ahora viene la parte que más tiempo consume: verificar el código fuente de la versión de Android que se utiliza. Dependiendo de la velocidad del disco duro y la conexión a Internet, este paso puede tomar varias horas, así que planifíquelo con anticipación. Además, tenga en cuenta que el código fuente es bastante grande, así que use una segunda partición con al menos 40 GB de espacio libre. Instalamos el código fuente para Android 4.4.4 de la siguiente manera:

```
user@lab:~$ mkdir ~/bin

user@lab:~$ PATH=~/bin:$PATH

user@lab:~$ curl https://storage.googleapis.com/git-repo-downloads/repo >
~/bin/repo

user@lab:~$ chmod a+x ~/bin/repo

user@lab:~$ repo init -u
https://android.googlesource.com/platform/ manifest -b android-
4.4.4_r1

user@lab:~$ repo sync
```

Después de haber instalado el código fuente para Android 4.4.4, ahora necesitamos las fuentes para el kernel que se ejecuta en el dispositivo en cuestión. Para el Nexus 4 que estamos usando aquí, el kernel correcto es el kernel de **mako**. Se puede encontrar una lista de todos los kernels disponibles para teléfonos Google en http://source.android.com/source/building-kernels.html.

```
user@lab:~$ git clone
https://android.googlesource.com/device/lge/mako- kernel/kernel

user@lab:~$ git clone https://android.googlesource.com/kernel/msm.git
```

Ahora que tenemos todas las fuentes necesarias para la compilación cruzada de LiME, es hora de obtener LiME en sí mismo:

```
user@lab:~$ git clone https://github.com/504ensicsLabs/LiME.git
```

Después de clonar el repositorio git en nuestra máquina de laboratorio, ahora tenemos que establecer algunas variables de entorno que se necesitan durante el proceso de compilación:

```
user@lab:~$ export SDK_PATH=/path/to/android-sdk-linux/

user@lab:~$ export NDK_PATH=/path/to/android-ndk/
```

```
user@lab:~$ export KSRC_PATH=/path/to/kernel-source/

user@lab:~$ export CC_PATH=$NDK_PATH/toolchains/arm-linux-
androideabi-4.9/prebuilt/linux-x86/bin/

user@lab:~$ export LIME_SRC=/path/to/lime/src
```

A continuación, necesitamos obtener la configuración actual del kernel del dispositivo en cuestión y copiarla a la ubicación correcta en la fuente de LiME. En nuestro Nexus 4, esto es posible al ingresar el siguiente comando:

```
user@lab:~$ adb pull /proc/config.gz

user@lab:~$ gunzip ./config.gz

user@lab:~$ cp config $KSRC_PATH/.config

user@lab:~$ cd $KSRC_PATH

user@lab:~$ make ARCH=arm CROSS_COMPILE=$CC_PATH/arm-eabi-modules_prepare
```

Antes de que podamos construir el módulo del kernel de LiME , necesitamos escribir nuestro Makefile personalizado:

```
obj-m := lime.o
lime-objs := main.o tcp.o disk.o
KDIR := /path/to/kernel-source
PWD := $(shell pwd)
CCPATH := /path/to/android-ndk/toolchains/arm-linux-androideabi-
4.4.4/ prebuilt/linux-x86/bin/
default:
 $(MAKE) ARCH=arm CROSS_COMPILE=$(CCPATH)/arm-eabi- -C $(KDIR)
M=$(PWD) modules
```

Con la ayuda de este Makefile, podemos construir el módulo kernel que se necesita para obtener la memoria volátil de un dispositivo Android. Ingresar `make` puede comenzar este proceso.

En el siguiente ejemplo, demostraremos cómo impulsar nuestro módulo kernel recién generado al dispositivo en cuestión y volcar toda la memoria volátil a nuestro entorno de laboratorio a través de TCP.

Si tiene un dispositivo en el que el kernel no permite cargar módulos sobre la marcha, debería considerar crear su propia imagen de recuperación (por ejemplo, una versión personalizada de TWRP o CWM), incluir el módulo de kernel LiME y flashearlo al dispositivo en cuestión. Si es lo suficientemente rápido durante la operación de flasheo, casi no se pierden datos (para obtener más información, consulte https://www1.informatik.uni-erlangen.de/frost).

El módulo LiME ofrece tres formatos de imagen diferentes que se pueden usar para guardar una imagen de memoria capturada en el disco: raw, padded, y lime. El tercer formato, lime, se analiza en detalle, ya que es nuestro formato de elección. El formato lime ha sido especialmente desarrollado para ser utilizado en conjunto con Volatility. Se supone que permite un análisis sencillo con Volatility y se ha agregado un espacio de direcciones especial para tratar este formato. Cada volcado de memoria que se basa en el formato lime tiene un encabezado de tamaño fijo, que contiene información específica de espacio de direcciones para cada rango de memoria. Esto elimina la necesidad de contar con rellenos adicionales sólo para llenar las regiones sin asignar o E/S mapeada en memoria. La especificación del encabezado LiME se enumera a continuación:

```
typedef struct {
  unsigned int magic;          // Always 0x4C694D45 (LiME)
  unsigned int version;          // Header version number
  unsigned long long s_addr;      // Starting address of physical RAM
  unsigned long long e_addr;      // Ending address of physical RAM
  unsigned char reserved[8];      // Currently all zeros
  }   attribute__ ((  packed  )) lime_mem_range_header;
```

Para obtener un archivo de ese tipo del dispositivo Android en cuestión, conéctese al dispositivo Android a través de `adb` e ingrese los siguientes comandos:

```
user@lab:~$ adb push lime.ko /sdcard/lime.ko
user@lab:~$ adb forward tcp:4444 tcp:4444
user@lab:~$ adb shell
nexus4:~$ su
nexus4:~$ insmod /sdcard/lime.ko "path=tcp:4444 format=lime"
```

En la máquina de laboratorio, ingrese el siguiente comando para aceptar los datos enviados a través del puerto TCP 4444 desde el dispositivo Android a la máquina de laboratorio local:

```
user@lab:~$ nc localhost 4444 > nexus4_ram.lime
```

Si los comandos anteriores se ejecutan con éxito, ahora tendrá un volcado de RAM que se puede analizar con la ayuda de Volatility u otras herramientas (consulte la siguiente sección).

Volatility para Android

Después de adquirir un archivo de volcado que representa la memoria física del sistema de destino con las herramientas que creamos en la sección anterior, tenemos la intención de extraer los artefactos de datos de él. Sin un análisis en profundidad de las estructuras de memoria de Android, solo podríamos extraer formatos de archivo conocidos como JPEG, o solo los encabezados JPEG con los datos EXIF (con herramientas como **PhotoRec**) o simples cadenas ASCII, que se almacenan en una forma contigua (con herramientas comunes de Linux como **strings**) que podría usarse para ataques de fuerza bruta en los dispositivos en cuestión. Este enfoque es muy limitado, ya que puede utilizarse para cualquier disco o volcado de memoria, pero no se centra en el sistema operativo y las estructuras específicas de la aplicación. Como intentamos extraer objetos de datos completos del sistema Android, haremos uso del popular framework de investigación forense para la memoria volátil: **Volatility**.

En esta sección, utilizaremos una versión de Volatility con soporte ARM (necesita la versión 2.3 como mínimo). Dada una imagen de memoria, Volatility puede extraer procesos en ejecución, abrir sockets de red, mapas de memoria para cada proceso y módulos kernel.

> Antes de poder analizar una imagen de memoria, se debe crear un perfil de Volatility que se pasa al framework Volatility como un parámetro de línea de comando. Dicho perfil de Volatility es un conjunto de definiciones de **vtype** y direcciones opcionales de símbolos que Volatility usa para localizar información sensible y analizarla.

Básicamente, un perfil es un archivo comprimido que contiene dos archivos, de la siguiente manera:

- El archivo System.map contiene nombres de símbolos y direcciones de estructuras de datos estáticos en el kernel de Linux. En el caso de Android, este archivo se encuentra en el árbol fuente del kernel después de la compilación del kernel.
- El archivo module.dwarf surge al compilar un módulo contra el kernel objetivo y extraer la información de depuración DWARF.

Para crear un archivo module.dwarf, se requiere una utilidad llamada dwarfdump. El árbol fuente de Volatility contiene el directorio tools/linux. Si ejecuta `make` en este directorio, el comando compila el módulo y produce el archivo DWARF deseado. La creación del perfil real se realiza simplemente ejecutando el siguiente comando:

```
user@lab $ zip Nexus4.zip module.dwarf System.map
```

El archivo ZIP resultante debe copiarse en volatility/plugins/overlays/linux. Después de copiar con éxito el archivo, el perfil aparece en la sección de perfiles de salida de ayuda de Volatility.

Aunque el soporte de Android en Volatility es bastante nuevo, existe una gran cantidad de plugins de Linux que también funcionan perfectamente en Android. Por ejemplo:

- `linux_pslist`: Enumera todos los procesos en ejecución de un sistema similar al comando ps de Linux
- `linux_ifconfig`: Este plugin simula el comando de Linux ifconfig
- `linux_route_cache`: Lee e imprime el caché de ruta que almacena las entradas de enrutamiento usadas recientemente en una tabla hash
- `linux_proc_maps`: Este complemento adquiere mapeos de memoria de cada proceso individual

Si está interesado en cómo escribir plugins de Volatility personalizados y analizar estructuras desconocidas en la Dalvik Virtual Machine (DVM), consulte el siguiente artículo escrito por mí y mis colegas: Análisis de memoria post-Mortem de dispositivos Android iniciados en frío (consulte https://www1.informatik.uni-erlangen.de/filepool/publications/android.ram.analysis.pdf).

En la siguiente sección, vamos a mostrar cómo reconstruir los datos de aplicaciones específicas con la ayuda de LiME y Volatility.

Reconstruyendo datos para Android

Ahora, veremos cómo reconstruir los datos de las aplicaciones con la ayuda de Volatility y los plugins personalizados. Por lo tanto, hemos elegido el historial de llamadas y la memoria caché del teclado. Si está investigando en un sistema Linux o Windows común, ya hay una gran cantidad de complementos disponibles, como verá en la última sección de este capítulo. Desafortunadamente, en Android, debes escribir tus propios complementos.

Historial de llamadas

Uno de nuestros objetivos es recuperar la lista de llamadas recientes entrantes y salientes desde un volcado de memoria de Android. Esta lista se carga cuando se abre la aplicación del teléfono. El proceso responsable para la aplicación del teléfono y el historial de llamadas es com.android.contacts. Este proceso carga el archivo de clase PhoneClassDetails.java que modela los datos de todas las llamadas telefónicas en una estructura de historial. Una instancia de esta clase está en memoria por entrada de historial. Los campos de datos para cada instancia son metainformación típica de una llamada, de la siguiente manera:

- Tipo (entrantes, salientes o perdidas)
- Duración
- Fecha y hora

- Número de teléfono
- Nombre de contacto
- Foto asignada del contacto

Para extraer y visualizar automáticamente estos metadatos, proporcionamos un plugin de Volatility llamado `dalvik_app_calllog`, que se muestra de la siguiente manera:

```
class dalvik_app_calllog(linux_common.AbstractLinuxCommand):

     def   init  (self, config, *args, **kwargs):
          linux_common.AbstractLinuxCommand.  init  (self, config,
*args, **kwargs)
          dalvik.register_option_PID(self._config)
          dalvik.register_option_GDVM_OFFSET(self._config)
          self._config.add_option('CLASS_OFFSET', short_option =
          'c',
default = None,
          help = 'This is the offset (in hex) of system class
PhoneCallDetails.java', action = 'store', type = 'str')

     def calculate(self):
          # if no gDvm object offset was specified, use this one
          if not self._config.GDVM_OFFSET:
               self._config.GDVM_OFFSET = str(hex(0x41b0))

          # use linux_pslist plugin to find process address space and
ID if not specified
          proc_as = None
          tasks =
          linux_pslist.linux_pslist(self._config).calculate() for
          task in tasks:
               if str(task.comm) == "ndroid.contacts":
                    proc_as = task.get_process_address_space()
                    if not self._config.PID:
                         self._config.PID = str(task.pid)
                    break

          # use dalvik_loaded_classes plugin to find class offset
if not specified
          if not self._config.CLASS_OFFSET:
              classes = dalvik_loaded_classes.dalvik_loaded_
classes(self._config).calculate()
              for task, clazz in classes:
```

```
                    if (dalvik.getString(clazz.sourceFile)+"" ==
"PhoneCallDetails.java"):
                        self._config.CLASS_OFFSET = str(hex(clazz.
obj_offset))
                        break

          # use dalvik_find_class_instance plugin to find a list of
possible class instances
          instances = dalvik_find_class_instance.dalvik_find_class_
instance(self._config).calculate()
          for sysClass, inst in instances:
               callDetailsObj = obj.Object('PhoneCallDetails', offset
= inst, vm = proc_as)
               # access type ID field for sanity check
               typeID = int(callDetailsObj.callTypes.contents0)
               # valid type ID must be 1,2 or 3
               if (typeID == 1 or typeID == 2 or typeID == 3):
                    yield callDetailsObj

     def render_text(self, outfd, data):
          self.table_header(outfd, [    ("InstanceClass", "13"),
                                        ("Date", "19"),
                                        ("Contact", "20"),
                                        ("Number", "15"),
                                        ("Duration", "13"),
                                        ("Iso", "3"),
                                        ("Geocode", "15"),
                                        ("Type", "8")
                                        ])
          for callDetailsObj in data:
               # convert epoch time to human readable date and time
               rawDate = callDetailsObj.date / 1000
               date =    str(time.gmtime(rawDate).tm_mday) + "." + \
                         str(time.gmtime(rawDate).tm_mon) + "." + \
                         str(time.gmtime(rawDate).tm_year) + " " + \
                         str(time.gmtime(rawDate).tm_hour) + ":" + \
                         str(time.gmtime(rawDate).tm_min) + ":" + \
                         str(time.gmtime(rawDate).tm_sec)

               # convert duration from seconds to hh:mm:ss format
               duration =     str(callDetailsObj.duration / 3600) + "h
" + \
```

```
                              str((callDetailsObj.duration % 3600) /
60) + "min " + \
                              str(callDetailsObj.duration % 60) + "s"

               # replace call type ID by string
               callType = int(callDetailsObj.callTypes.contents0)
               if callType == 1:
                    callType = "incoming"
               elif callType == 2:
                    callType = "outgoing"
               elif callType == 3:
                    callType = "missed"
               else:
                    callType = "unknown"

               self.table_row(     outfd,
                                   hex(callDetailsObj.obj_offset),
                                   date,
                                   dalvik.parseJavaLangString(callDeta
ilsObj.name.dereference_as('StringObject')),
                                   dalvik.parseJavaLangString(callDeta
ilsObj.formattedNumber.dereference_as('StringObject')),
                                   duration,
                                   dalvik.parseJavaLangString(callDeta
ilsObj.countryIso.dereference_as('StringObject')),
                                   dalvik.parseJavaLangString(callDeta
ilsObj.geoCode.dereference_as('StringObject')),
                                   callType)
```

Este plugin acepta los siguientes parámetros de línea de comando:

- `-o`: Para un desplazamiento al objeto gDvm
- `-p`: Para un process ID (PID)
- `-c`: Para un desplazamiento a la clase PhoneClassDetails

Si se conocen algunos de estos parámetros y se pasan al plugin, el tiempo de ejecución del plugin se reduce significativamente. De lo contrario, el plugin debe buscar estos valores en la RAM.

Caché del teclado

Ahora, queremos echar un vistazo a la memoria caché de la aplicación de teclado predeterminada. Suponiendo que no se dieron más entradas después de desbloquear la pantalla y el teléfono inteligente está protegido por un PIN, este PIN es igual a la última entrada del usuario, que se puede encontrar en un volcado de memoria Android como una cadena Unicode UTF-16. La cadena Unicode de la última entrada del usuario la crea la clase `RichInputConnection` en el proceso `com.android.inputmethod.latin` y se almacena en una variable llamada `mCommittedTextBeforeComposingText`. Esta variable es como un búfer de teclado, es decir, almacena las últimas pulsaciones de teclas confirmadas y mecanografiadas del teclado en pantalla. Para recuperar la última entrada de usuario, proporcionamos un plugin de Volatility llamado `dalvik_app_lastInput`, de la siguiente manera:

```
class dalvik_app_lastInput(linux_common.AbstractLinuxCommand):

     def   init  (self, config, *args, **kwargs):
          linux_common.AbstractLinuxCommand.  init  (self, config,
*args, **kwargs)
          dalvik.register_option_PID(self._config)
          dalvik.register_option_GDVM_OFFSET(self._config)
          self._config.add_option('CLASS_OFFSET', short_option =
          'c',
default = None,
          help = 'This is the offset (in hex) of system class
RichInputConnection.java', action = 'store', type = 'str')

     def calculate(self):

          # if no gDvm object offset was specified, use this one
          if not self._config.GDVM_OFFSET:
               self._config.GDVM_OFFSET = str(0x41b0)

          # use linux_pslist plugin to find process address space and
ID if not specified
          proc_as = None
          tasks =
          linux_pslist.linux_pslist(self._config).calculate() for
          task in tasks:
               if str(task.comm) == "putmethod.latin":
                    proc_as = task.get_process_address_space()
                    self._config.PID = str(task.pid)
                    break

          # use dalvik_loaded_classes plugin to find class offset
if not specified
          if not self._config.CLASS_OFFSET:
```

```
            classes = dalvik_loaded_classes.dalvik_loaded_
classes(self._config).calculate()
            for task, clazz in classes:
                if (dalvik.getString(clazz.sourceFile)+"" ==
"RichInputConnection.java"):
                    self._config.CLASS_OFFSET = str(hex(clazz.
obj_offset))
                    break

        # use dalvik_find_class_instance plugin to find a list of
possible class instances
        instance = dalvik_find_class_instance.dalvik_find_class_
instance(self._config).calculate()
        for sysClass, inst in instance:
            # get stringBuilder object
            stringBuilder = inst.clazz.getJValuebyName(inst,
"mCommittedTextBeforeComposingText").Object.dereference_as('Object')
            # get superclass object
            abstractStringBuilder = stringBuilder.clazz.super.
dereference_as('ClassObject')

            # array object of super class
            charArray = abstractStringBuilder.
getJValuebyName(stringBuilder, "value").Object.dereference_
as('ArrayObject')
            # get length of array object
            count = charArray.length
            # create string object with content of the array object
            text = obj.Object('String', offset = charArray.
contents0.obj_offset,
            vm = abstractStringBuilder.obj_vm, length = count*2,
encoding = "utf16")
            yield inst, text

    def render_text(self, outfd, data):
        self.table_header(outfd, [    ("InstanceClass", "13"),
                                      ("lastInput", "20")
                                      ])
        for inst, text in data:

            self.table_row(     outfd,
                                hex(inst.obj_offset),
                                text)
```

En realidad, este plugin no solo recupera los PIN sino también las entradas de usuario arbitrarias que se dieron por última vez; esto podría ser un artefacto interesante de evidencia digital en muchos casos. De forma similar al plugin anterior, acepta los mismos tres parámetros de línea de comando: desplazamiento de gDvm, PID y desplazamiento de archivo de clase. Si ninguno, o solo algunos de estos parámetros se dan, el complemento también puede determinar automáticamente los valores perdidos.

Usando Volatility en Linux

En la siguiente sección, describiremos técnicas de adquisición de memoria y casos de uso de muestra para usar Volatility para el análisis forense de memoria de Linux.

Adquisición de memoria

Si el sistema no está virtualizado y, por lo tanto, no hay forma de obtener la memoria directamente desde la capa del hipervisor; entonces, incluso para Linux, nuestra herramienta de elección es LiME.

Sin embargo, a diferencia de Android, la instalación y operación de la herramienta es mucho más fácil porque generamos y ejecutamos LiME directamente en el sistema Linux; sin embargo, muchos pasos son bastante similares, como notará en los párrafos siguientes.

Primero, determine la versión exacta del kernel, que se está ejecutando en el sistema, que se analizará. Si no hay suficiente documentación disponible, puede ejecutar el siguiente comando para obtener la versión del kernel:

```
user@forensic-target $ uname –a
```

```
Linux forensic-target 3.2.0-88-generic #126-Ubuntu SMP Mon Jul 6
21:33:03 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
```



Use la gestión de configuración en entornos empresariales

Los entornos empresariales a menudo ejecutan sistemas de gestión de configuración que le muestran la versión del kernel y la distribución de Linux de su sistema destino. Pidale a su cliente que le proporcione esta información o incluso un sistema con una versión de kernel y un entorno de software idénticos puede ayudarlo a reducir el riesgo de incompatibilidades entre el módulo LiME y su objetivo forense.

En su entorno de laboratorio, prepare el módulo del kernel LiME para la adquisición de memoria. Para compilar el módulo, asegúrese de tener la versión correcta del código fuente del kernel disponible para su objetivo y luego emita el siguiente comando de compilación en el directorio src de LiME:

```
user@lab src $ make -C /usr/src/linux-headers-3.2.0-88-generic M=$PWD
```

Esto debería crear el módulo`lime.ko` en el directorio actual.

En el sistema de destino, este módulo de kernel se puede utilizar para volcar la memoria en el disco, de la siguiente manera:

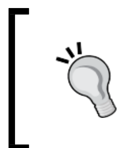
```
user@forensic-target $ sudo insmod lime.ko
path=/path/to/dump.lime format=lime
```



Recomendamos elegir una ruta en la red para escribir la imagen. De esta manera, los cambios realizados en el sistema local son mínimos. Transferir la imagen a través de la red también es una opción. Simplemente siga la descripción en la sección Uso de Volatility en Android.

Volatility para Linux

Volatility viene con una amplia gama de *perfiles*. Volatility usa estos perfiles para interpretar el volcado de memoria. Desafortunadamente, la gran variedad de kernels de Linux, arquitecturas de sistema y configuraciones de kernel hacen que sea imposible enviar los perfiles de todas las versiones de kernels de Linux.



Listado de todos los perfiles de Volatility
La lista de todos los perfiles disponibles se puede recuperar con el comando `vol.py --info`.

En consecuencia, puede ser necesario crear su propio perfil como una pareja ideal para el objetivo forense. El framework Volatility admite este paso al proporcionar un módulo ficticio que se debe compilar contra los encabezados del kernel del sistema de destino. Este módulo está disponible en la distribución Volatility en el subdirectorio `tools/linux`. Compilarlo, similar a LiME, pero con la configuración de depuración habilitada:

```
user@lab src $ make -C /usr/src/linux-headers-3.2.0-88-generic
CONFIG_ DEBUG_INFO=y M=$PWD
```

Esto crea `module.ko`. No hay necesidad de cargar este módulo; todo lo que necesitamos es su información de depuración. Usamos la herramienta `dwarfdump`, que está disponible como un paquete de instalación en la mayoría de las distribuciones de Linux, para extraer esta información de depuración:

```
user@lab $ dwarfdump -di module.ko > module.dwarf
```

El siguiente paso en la creación de nuestro perfil es adquirir el archivo `System.map` del sistema de destino o un sistema con la misma arquitectura, la versión del kernel y la configuración del kernel. El archivo `System.map` se puede encontrar en el directorio `/boot`. A menudo, la versión del kernel se incluye en el nombre del archivo, por lo tanto, asegúrese de seleccionar el archivo `System.map` para el kernel en ejecución del sistema de destino forense.

Coloque `module.dwarf` y `System.map` en un archivo zip, que se convertirá en nuestro perfil de Volatility, como se muestra a continuación:

```
user@lab $ zip Ubuntu3.2.0-88.zip module.dwarf System.map
```

Como se muestra en el ejemplo, el nombre del archivo ZIP debe reflejar la distribución y la versión del kernel.

> Asegúrese de no agregar información de ruta adicional al archivo zip. De lo contrario, Volatility puede no cargar los datos del perfil.

Copie el nuevo perfil en el directorio de perfil de Linux de Volatility, de la siguiente manera:

```
user@lab $ sudo cp Ubuntu3.2.0-88.zip /usr/local/lib/python2.7/dist-
packages/volatility-2.4-py2.7.egg/volatility/plugins/overlays/linux/
```

En lugar de utilizar el directorio de perfiles de todo el sistema, también puede elegir uno nuevo y agregar la opción `--plugins=/path/to/profiles` a su línea de comando Volatility.

Finalmente, debe obtener el nombre de su nuevo perfil para un uso posterior. Por lo tanto, use la siguiente llamada:

```
user@lab $ vol.py --info
```

La salida debe contener una línea adicional que muestre el nuevo perfil, como se muestra a continuación:

```
Profiles
--------
LinuxUbuntu3_2_0-88x64 - A Profile for Linux Ubuntu3.2.0-88 x64
```

Para usar este perfil, agregue `--profile = LinuxUbuntu3_2_0-88x64` como el argumento de línea de comando para todas las llamadas posteriores a `vol.py`.

Reconstrucción de datos para Linux

Todos los plugins que analizan los volcados de memoria de Linux tienen el prefijo `linux_`. Por lo tanto, debe usar la versión de Linux de los plugins. De lo contrario, puede recibir un mensaje de error que notifique que el módulo no es compatible con el perfil seleccionado.

Analizando procesos y módulos

Un primer paso típico en el análisis de un volcado de memoria es enumerar todos los procesos en ejecución y los módulos de kernel cargados.

A continuación, se muestra cómo separar todos los procesos en ejecución de un volcado de memoria con Volatility:

```
user@lab $ vol.py --profile=LinuxUbuntu3_2_0-88x64 --
file=memDump.lime linux_pslist

Volatility Foundation Volatility Framework 2.4


Offset             Name                 Pid             Uid
Gid    DTB                  Start Time

------------------ -------------------- --------------- ---------------
------ ------------------ ----------

0xffff8802320e8000  init                 1               0               0
0x000000022f6c0000  2015-08-16 09:51:21 UTC+0000

0xffff8802320e9700 kthreadd             2               0               0
------------------ 2015-08-16 09:51:21 UTC+0000


0xffff88022fbc0000  cron                 2500            0               0
0x000000022cd38000  2015-08-16 09:51:25 UTC+0000

0xffff88022fbc1700  atd                  2501            0               0
0x000000022fe28000  2015-08-16 09:51:25 UTC+0000

0xffff88022f012e00  irqbalance           2520            0               0
0x000000022df39000  2015-08-16 09:51:25 UTC+0000

0xffff8802314b5c00 whoopsie             2524            105
114    0x000000022f1b0000 2015-08-16 09:51:25 UTC+0000

0xffff88022c5c0000 freshclam            2598            119
131    0x0000000231fa7000 2015-08-16 09:51:25 UTC+0000
```

Como se muestra en el resultado, el plugin `linux_pslist` itera la estructura del kernel al describir los procesos activos, es decir, comienza desde el símbolo `init_task` e itera la lista vinculada `task_struct-> tasks`. El plugin obtiene una lista de todos los procesos en ejecución, incluida su dirección de desplazamiento en la memoria, nombre del proceso, ID de proceso (PID), ID numérico del usuario y grupo del proceso (UID y GID) y hora de inicio. **La Base de tabla de directorios (DTB)** se puede usar en el análisis posterior para traducir direcciones físicas a direcciones virtuales. Las entradas vacías de DTB se relacionan, muy probablemente, con un hilo del kernel. Por ejemplo, `kthreadd` en nuestra salida de ejemplo.

Analizar la información de la red

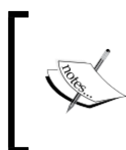
El volcado de memoria contiene diversa información sobre la actividad de la red de nuestro sistema de destino forense. Los siguientes ejemplos muestran cómo utilizar Volatility para derivar la información sobre la actividad de red reciente.

El caché del **protocolo de resolución de direcciones (ARP)** del kernel de Linux asigna las direcciones MAC a las direcciones IP. Antes de que se establezca una comunicación de red en la red local, el kernel de Linux envía una solicitud ARP para obtener la información sobre la dirección MAC correspondiente para una dirección IP de destino determinada. La respuesta se almacena en caché, en la memoria para su reutilización para comunicarse aún más con esta dirección IP en la red local. En consecuencia, las entradas de la memoria caché ARP indican los sistemas en la red local con los que el objetivo forense se estaba comunicando.

Para leer la memoria caché ARP desde un volcado de memoria Linux, use el siguiente comando:

```
user@lab $ vol.py --profile=LinuxUbuntu3_2_0-88x64 --
file=memDump.lime linux_arp
[192.168.167.22                         ] at 00:00:00:00:00:00    on
eth0
[192.168.167.20                         ] at b8:27:eb:01:c2:8f    on
eth0
```

Este extracto del resultado muestra que el sistema tenía una entrada de caché para la dirección de destino `192.168.167.20` con `b8:27:eb:01:c2:8f` siendo la dirección MAC correspondiente. La primera entrada es muy probablemente una entrada de caché que resulta de un intento de comunicación fallido, es decir, el socio de comunicación `192.168.167.22` no envió una respuesta a una solicitud ARP que se transmitió desde el sistema y, por lo tanto, la entrada de caché ARP correspondiente permaneció en su valor inicial de `00:00:00:00:00:00`. O el interlocutor de comunicación no era accesible o simplemente no existe.



Si grandes partes de la subred local aparecen en la memoria caché ARP con múltiples entradas que tienen una dirección MAC de 00: 00: 00: 00: 00: 00, entonces este es un indicador de la actividad de escaneo, es decir, el sistema ha intentado detectar otros sistemas en la red local.

Para un análisis de red adicional, podría valer la pena consultar la lista de direcciones MAC que se recuperan de la memoria caché ARP contra los sistemas que se supone que están en la subred local. Si bien esta técnica no es a prueba de balas (ya que las direcciones MAC se pueden falsificar), podría ayudar a descubrir dispositivos de red deshonestos.



Buscando el proveedor de hardware para una dirección MAC

El prefijo de una dirección MAC revela el proveedor de hardware del hardware de red correspondiente. Sitios como http: // www.macvendorlookup.com proporciona una indicación del proveedor de hardware de una tarjeta de red.

Si buscamos el proveedor de hardware para la dirección MAC b8:27:eb:01:c2:8f de nuestro ejemplo, muestra que este dispositivo fue fabricado por la Fundación Raspberry Pi. En un entorno estándar de oficina o centro de datos, estos dispositivos integrados rara vez se utilizan y, definitivamente, vale la pena comprobar si este dispositivo es benigno.

Para obtener una visión general de la actividad de la red en el momento en que se creó el volcado de memoria, Volatility proporciona los medios para emular el comando linux_netstat, de la siguiente manera:

```
user@lab $ vol.py --profile=LinuxUbuntu3_2_0-88x64 --
file=memDump.lime linux_netstat

TCP      192.168.167.21  :55622 109.234.207.112  :  143 ESTABLISHED
thunderbird/3746

UNIX 25129          thunderbird/3746

TCP      0.0.0.0         : 7802 0.0.0.0         :    0 LISTEN
skype/3833
```

Estas tres líneas son solo un pequeño extracto de la salida típica de este comando. La primera línea muestra que el proceso *Thunderbird* con el PID 3746 tiene una conexión de red ESTABLISHED activa al servidor IMAP (puerto TCP 143) con la dirección IP 109.234.207.112. La segunda línea simplemente muestra un socket de tipo UNIX que se utiliza para **comunicación entre procesos (IPC)**. La última entrada muestra que *Skype* con el 3833 PID es un LISTEN de espera para las conexiones entrantes en el puerto TCP 7802.

Volatility también se puede utilizar para reducir la lista de procesos a aquellos con acceso de red sin formato. Normalmente, este tipo de acceso solo es necesario para los clientes de **Protocolo de configuración dinámica de host (DHCP)**, diagnósticos de red y, por supuesto, malware para construir paquetes arbitrarios en la interfaz de red, por ejemplo, realizar un ataque llamado envenenamiento de caché ARP. A continuación, se muestra cómo enumerar los procesos con sockets de red sin formato:

```
user@lab $ vol.py --profile=LinuxUbuntu3_2_0-88x64 --
file=memDump.lime linux_list_raw
Process          PID    File Descriptor Inode
---------------- ------ --------------- ------------------
dhclient           2817               5            15831
```

Aquí, solo se detecta que el cliente DHCP tiene el acceso de red sin formato.



Detección de módulos Rootkit

Volatility proporciona una variedad de mecanismos para detectar el comportamiento típico del rootkit, por ejemplo, el enganche de interrupción, las manipulaciones de la pila de red y los módulos ocultos del kernel. Recomendamos familiarizarse con estos módulos ya que pueden acelerar su análisis. Además, revise periódicamente las actualizaciones de módulos para aprovechar los nuevos mecanismos de detección de malware que se incorporan a Volatility.

Algunos métodos genéricos y heurísticos para la detección de malware se combinan en el módulo *linux_malfind*. Este módulo busca mapeos de memoria de procesos sospechosos y produce una lista de procesos posiblemente maliciosos.

Caza de malware con la ayuda de YARA

YARA en sí es una herramienta que puede hacer coincidir un patrón dado en archivos y conjuntos de datos arbitrarios. Las reglas correspondientes, también conocidas como firmas, son una gran manera de buscar archivos maliciosos conocidos en volcados de discos duros o memoria.

En esta sección, queremos demostrar cómo buscar malware dado en un volcado de memoria adquirido de una máquina Linux. Por lo tanto, puede usar dos procedimientos diferentes que discutiremos a continuación:

- Buscando en el volcado de memoria directamente con la ayuda de YARA
- Usando *linux_yarascan* y *Volatility*

La primera opción tiene una gran desventaja; como ya sabemos, los volcados de memoria contienen datos fragmentados que normalmente son contiguos. Este hecho lo hace propenso al fracaso si está buscando en este volcado firmas conocidas, ya que no están necesariamente en el orden en que las está buscando.

La segunda opción, que utiliza *linux_yarascan*, es más segura ya que usa Volatility y conoce la estructura del volcado de memoria adquirido. Con la ayuda de este conocimiento, es capaz de resolver la fragmentación y realizar la busqueda confiable para firmas conocidas. Aunque, estamos usando *linux_yarascan* en Linux, este módulo también está disponible para *Windows (yarascan)* y *Mac OS X (mac_yarascan).*

Las principales capacidades de este módulo son las siguientes:

- Escanea los procesos dados en el volcado de memoria para una firma dada de YARA
- Escanea el rango completo de la memoria del kernel
- Extrae las áreas de memoria en el disco que contengan resultados positivos a las reglas YARA dadas

La lista completa de posibles opciones de línea de comando se puede ver al ingresar a `vol.py linux_yarascan -h`

Básicamente, puedes buscar de muchas maneras diferentes. La forma más sencilla de usar este módulo es buscar una URL determinada en el volcado de memoria. Esto puede hacerse ingresando el siguiente comando:

```
user@lab $ vol.py --profile=LinuxUbuntu3_2_0-88x64 --
file=memDump.lime linux_yarascan --yara-rules="microsoft.com" --wide


Task: skype pid 3833 rule r1 addr 0xe2be751f
0xe2be751f  6d 00 69 00 63 00 72 00 6f 00 73 00 6f 00 66 00
m.i.c.r.o.s.o.f.
0xe2be752f  74 00 2e 00 63 00 6f 00 6d 00 2f 00 74 00 79 00
t...c.o.m./.t.y.
0xe2be753f  70 00 6f 00 67 00 72 00 61 00 70 00 68 00 79 00
p.o.g.r.a.p.h.y.
0xe2be754f  2f 00 66 00 6f 00 6e 00 74 00 73 00 2f 00 59 00
/.f.o.n.t.s./.Y.
0xe2be755f  6f 00 75 00 20 00 6d 00 61 00 79 00 20 00 75 00
o.u...m.a.y...u.
0xe2be756f  73 00 65 00 20 00 74 00 68 00 69 00 73 00 20 00
s.e...t.h.i.s...
0xe2be757f  66 00 6f 00 6e 00 74 00 20 00 61 00 73 00 20 00
f.o.n.t...a.s...
```

```
0xe2be758f  70 00 65 00 72 00 6d 00 69 00 74 00 74 00 65 00
p.e.r.m.i.t.t.e.
0xe2be759f  64 00 20 00 62 00 79 00 20 00 74 00 68 00 65 00
d...b.y...t.h.e.
0xe2be75af  20 00 45 00 55 00 4c 00 41 00 20 00 66 00 6f 00
..E.U.L.A...f.o.
0xe2be75bf  72 00 20 00 74 00 68 00 65 00 20 00 70 00 72 00
r...t.h.e...p.r.
0xe2be75cf  6f 00 64 00 75 00 63 00 74 00 20 00 69 00 6e 00
o.d.u.c.t...i.n.
0xe2be75df  20 00 77 00 68 00 69 00 63 00 68 00 20 00 74 00
..w.h.i.c.h...t.
0xe2be75ef  68 00 69 00 73 00 20 00 66 00 6f 00 6e 00 74 00
h.i.s...f.o.n.t.
0xe2be75ff  20 00 69 00 73 00 20 00 69 00 6e 00 63 00 6c 00
..i.s...i.n.c.l.
0xe2be760f  75 00 64 00 65 00 64 00 20 00 74 00 6f 00 20 00
u.d.e.d...t.o...
Task: skype pid 3833 rule r1 addr 0xedfe1267
0xedfe1267  6d 00 69 00 63 00 72 00 6f 00 73 00 6f 00 66 00
m.i.c.r.o.s.o.f.
0xedfe1277  74 00 2e 00 63 00 6f 00 6d 00 2f 00 74 00 79 00
t...c.o.m./.t.y.
0xedfe1287  70 00 6f 00 67 00 72 00 61 00 70 00 68 00 79 00
p.o.g.r.a.p.h.y.
0xedfe1297  2f 00 66 00 6f 00 6e 00 74 00 73 00 2f 00 59 00
/.f.o.n.t.s./.Y.
0xedfe12a7  6f 00 75 00 20 00 6d 00 61 00 79 00 20 00 75 00
o.u...m.a.y...u.
0xedfe12b7  73 00 65 00 20 00 74 00 68 00 69 00 73 00 20 00
s.e...t.h.i.s...
0xedfe12c7  66 00 6f 00 6e 00 74 00 20 00 61 00 73 00 20 00
f.o.n.t...a.s...
0xedfe12d7  70 00 65 00 72 00 6d 00 69 00 74 00 74 00 65 00
p.e.r.m.i.t.t.e.
0xedfe12e7  64 00 20 00 62 00 79 00 20 00 74 00 68 00 65 00
d...b.y...t.h.e.
0xedfe12f7  20 00 45 00 55 00 4c 00 41 00 20 00 66 00 6f 00
..E.U.L.A...f.o.
0xedfe1307  72 00 20 00 74 00 68 00 65 00 20 00 70 00 72 00
r...t.h.e...p.r.
```

```
0xedfe1317  6f 00 64 00 75 00 63 00 74 00 20 00 69 00 6e 00
o.d.u.c.t...i.n.
0xedfe1327  20 00 77 00 68 00 69 00 63 00 68 00 20 00 74 00
..w.h.i.c.h...t.
0xedfe1337  68 00 69 00 73 00 20 00 66 00 6f 00 6e 00 74 00
h.i.s...f.o.n.t.
0xedfe1347  20 00 69 00 73 00 20 00 69 00 6e 00 63 00 6c 00
..i.s...i.n.c.l.
0xedfe1357  75 00 64 00 65 00 64 00 20 00 74 00 6f 00 20 00
u.d.e.d...t.o...
```

Una forma más compleja pero también más realista es buscar una regla dada de YARA. La siguiente regla de YARA fue hecha para identificar si un sistema ha sido infectado con la familia de malware *Derusbi*:

```
rule APT_Derusbi_Gen
{
meta:
  author = "ThreatConnect Intelligence Research Team"
strings:
  $2 = "273ce6-b29f-90d618c0" wide ascii
  $A = "Ace123dx" fullword wide ascii
  $A1 = "Ace123dxl!" fullword wide ascii
  $A2 = "Ace123dx!@#x" fullword wide ascii
  $C = "/Catelog/login1.asp" wide ascii
  $DF = "~DFTMP$$$$$.1" wide ascii
  $G = "GET /Query.asp?loginid=" wide ascii
  $L = "LoadConfigFromReg failded" wide ascii
  $L1 = "LoadConfigFromBuildin success" wide ascii
  $ph = "/photoe/photo.asp HTTP" wide ascii
  $PO = "POST /photos/photo.asp" wide ascii
  $PC = "PCC_IDENT" wide ascii
condition:
  any of them
}
```

Si guardamos esta regla como apt_derusbi_gen.rule, podemos buscarla en el volcado de memoria adquirido al ingresar el siguiente comando:

```
user@lab $ vol.py --profile=LinuxUbuntu3_2_0-88x64 --
file=memDump.lime linux_yarascan --yara-file=apt_derusbi_gen.rule --
wide
```

El resultado solo nos mostrará una vista previa corta que se puede ampliar utilizando la opción `--size`.

Si está investigando un escenario predefinido (por ejemplo, si ya sabe que el sistema ha sido atacado por un grupo conocido), puede copiar todas sus reglas en un único archivo de reglas y buscar en el volcado de memoria todas las reglas del archivo En seguida. Volatility y *linux_yarascan* mostrarán cada hit y su número de regla correspondiente. Esto hace que sea mucho más rápido buscar comportamiento malicioso conocido en un volcado de memoria.

Existe una gran cantidad de fuentes para las firmas YARA que están disponibles en la naturaleza y solo mencionaremos algunas de las más importantes aquí para ayudarlo, comenzando con la búsqueda de malware como se muestra a continuación:

- Grupo de intercambio de firmas YARA en Grupos de Google: `http:// www. deependresearch.org/`
- Firmas de AlienVault Labs:`https://github.com/AlienVault-Labs/AlienVaultLabs/tree/master/malware_analysis`
- Firmas de antivirus que pueden compilarse con la ayuda de ClamAV y la receta 3-3 de Malware Analyst's Cookbook: `https://code.google.com/p/malwarecookbook/source/browse/trunk/3/3/clamav_to_yara.py`

Resumen

En este capítulo, proporcionamos una descripción general de la memoria forense utilizando el framework Volatility. En los ejemplos, demostramos técnicas de adquisición de memoria para sistemas Android y Linux y vimos cómo usar LiME en ambos sistemas. Usamos Volatility para obtener información sobre procesos en ejecución, módulos cargados, posible actividad maliciosa y actividad de red reciente. Esto último es útil para rastrear las actividades de un atacante a través de la red.

En el último ejemplo de este capítulo, demostramos cómo buscar una determinada firma de malware u otras reglas altamente flexibles basadas en patrones en dicho volcado de memoria. Estas firmas y reglas de YARA ayudan a identificar actividades o archivos sospechosos muy rápido.

Además, demostramos cómo obtener la caché del teclado y el historial de llamadas desde un dispositivo Android.

A dónde ir desde aquí

Si desea probar las herramientas y el conocimiento obtenido de este libro, tenemos los siguientes dos consejos para usted:

- Cree un laboratorio con dos máquinas virtuales: **Metasploit** y **Metasploitable**. Intenta piratear tu sistema **Metasploitable** y luego realiza un análisis forense. ¿Eres capaz de reconstruir el ataque y reunir todos los Indicadores de compromiso?
- Obtenga algunos discos duros viejos, que ya no se usan pero que se han usado regularmente en el pasado. Realice un análisis forense en estas unidades y trate de reconstruir la mayor cantidad de datos posible. ¿Eres capaz de reconstruir operaciones anteriores en estas unidades?

Si desea mejorar su conocimiento sobre algunos de los temas tratados en este libro, los siguientes libros son una muy buena opción:

- *Práctica forense móvil por Satish Bommisetty, Rohit Tamma, Heather Mahalik, Packt Publishing*
- *El arte de la memoria forense: Detección de malware y amenazas en Windows, Linux y memoria Mac por Michael Hale Ligh, Andrew Case, Jamie Levy y AAron Walters, Wiley India*
- *Manual de análisis forense digital e Investigación por Eoghan Casey, Academic Press*

Index

E

F

G

H

I

W

Y



Thank you for buying

Mastering Python Forensics

About Packt Publishing

Packt, pronounced 'packed', published its first book, *Mastering phpMyAdmin for Effective MySQL Management*, in April 2004, and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern yet unique publishing company that focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website at `www.packtpub.com`.

About Packt Open Source

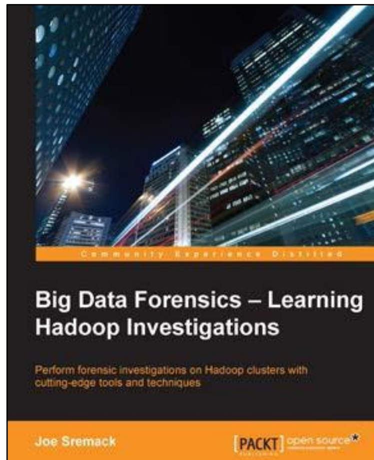
In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around open source licenses, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each open source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it fi before writing a formal book proposal, then please contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] open source
community experience distilled
PUBLISHING

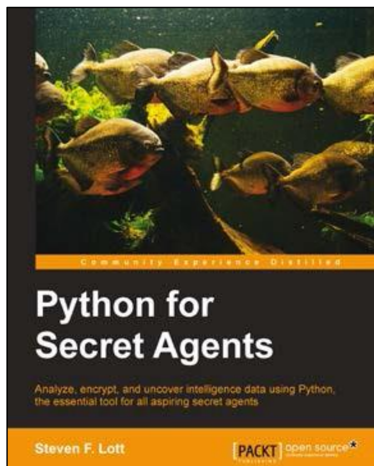


Big Data Forensics – Learning Hadoop Investigations

ISBN: 978-1-78528-810-4 Paperback: 264 pages

Perform forensic investigations on Hadoop clusters with cutting-edge tools and techniques

1. Identify, collect, and analyze Hadoop evidence forensically.
2. Learn about Hadoop's internals and Big Data file storage concepts.
3. A step-by-step guide to help you perform forensic analysis using freely available tools.



Python for Secret Agents

ISBN: 978-1-78398-042-0 Paperback: 216 pages

Analyze, encrypt, and uncover intelligence data using Python, the essential tool for all aspiring secret agents

1. Build a toolbox of Python gadgets for password recovery, currency conversion, and civic data hacking.
2. Use steganography to hide secret messages in images.
3. Get to grips with geocoding to find villains' secret lairs.

Please check **www.PacktPub.com** for information on our titles